Ian Sinclair

PROGRAMANDO CON AMSTRAD

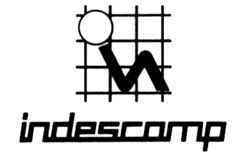


AMSTRAD

Programando con Amstrad CPC464

Programando con Amstrad CPC464

Ian Sinclair



PROGRAMANDO CON AMSTRAD CPC464

Edición española de la obra

AMSTRAD COMPUTING
Ian Sinclair

publicada en castellano bajo licencia de

Granada Technical Books
Granada Publishing Ltd
8 Grafton Street, London W1X 3LA

INDESCOMP, S. A.
Avda. del Mediterráneo, 9
28007 Madrid

ISBN 84 86176 23 9
Depósito Legal: M-2537-85

Impresión: Gráficas Lormo. Isabel Méndez, 15. Madrid.

Producción de la edición española:
Vector Ediciones
Gutierre de Cetina, 61
28017 Madrid

Contenido

Prólogo VII
1 Instalación del CPC464 1
2 Manejo de la pantalla 17
3 Variaciones sobre el mismo tema 29
4 Repetición de tareas 45
5 Cadenas literales 57
6 Menúes, subrutinas y programas 73
7 Archivos de datos en cinta 89
8 Ventanas y otros efectos 107
9 Introducción a los gráficos 119
10 Gráficos más avanzados 131
11 Los sonidos del CPC464 145
Apéndice A: Edición 167
Apéndice B: Conexión de una impresora 172
Apéndice C: Programación del teclado 173
Apéndice D: Trampas para errores 175
Índice 179

Prólogo

El ordenador Amstrad CPC464 ofrece muchas más prestaciones, tanto para el aficionado como para el profesional, que cualquier otra máquina de su mismo nivel de precio, más incluso que otros ordenadores de precio muy superior. Es muy probable que la mayor parte de sus usuarios no haya utilizado nunca un ordenador, y casi seguro que no uno tan avanzado como éste. El manual que se suministra con la máquina es uno de los mejores jamás publicados para un ordenador doméstico, pero es evidente que no puede responder a todas las necesidades. En particular, una cosa es saber utilizar un ordenador y otra muy distinta aprender a programarlo, y sobre todo a diseñar los propios programas. El objetivo de este libro es ayudar al usuario principiante en esa evolución gradual.

Por muchos que sean los programas que se pongan a la venta para este ordenador, inevitablemente habrá alguna aplicación que no sea cubierta por ellos. Cuando los conocimientos del lector hayan alcanzado la fase en la que sea capaz de diseñar los programas adecuados para resolver sus propios problemas, necesitará saber como programar el Amstrad CPC464. La programaciónn es, después de todo, la principal tarea para la que se compra un ordenador. Tener un ordenador y no programarlo es como comprar un Mercedes y hacer que lo conduzca un chófer. Si el lector no ha programado nunca un ordenador, éste libro le enseñará a hacerlo. En cambio, si ha conocido otros ordenadores, este libro le dará la clave de muchas posibilidades que son exclusivas del Amstrad. El BASIC del Amstrad CPC464 es muy diferente del de otras máquinas anteriores, tanto que es necesario aprenderlo partiendo de cero.

Debo resaltar que he escrito este libro *utilizando* un Amstrad CPC464, y que los programas que ofrezco como ejemplos han sido comprobados en él. Esta afirmación puede aparecer innecesaria, pero es un hecho que se publican muchos libros cuyos programas *no funcionan*. Todas las afirmaciones de este libro han sido comprobadas; no he copiado nada del manual sin la adecuada comprobación. En los casos en los que una orden o una instrucción no funcionan tal como el manual asegura, he hecho notar la diferencia. El libro lo he desarrollado utilizando un monitor en color, pues creo que muchos de los lectores habrán comprado o comprarán el ordenador con ese monitor, dado el precio notablemente bajo que tiene. No obstante, no he olvidado a los lectores que hayan comprado el ordenador con la fuente de alimentación y adaptador para TV.

Como siempre, estoy muy agradecido a las muchas personas que han hecho posible este libro. La máquina me fue enviada por AMSTRAD CONSUMER ELECTRONICS PLC; Richard Miles, de COLLINS PROFESSIONAL AND TECHNICAL BOOKS, trabajó incansablemente para conseguir que el ordenador estuviera a mi disposición lo más pronto posible. Tanto él como Janet Murphy, además de otros miembros del personal de Collins Professional and Technical Books, han hecho maravillas con el manuscrito. Los teclistas de fotocomposición y los impresores, los más eficaces que conozco, han conseguido que el libro viera la luz en un tiempo record. Espero que toda esta labor de equipo haya dado como resultado un libro que esté a la altura de la calidad del ordenador al que va dedicado. El Amstrad CPC464 es uno de los mejores ordenadores que he conocido; se puede felicitar a Amstrad por el acierto con que han debutado en el mercado informático. Ha sido una experiencia muy grata descubrir un ordenador en el que todo funciona según dice el fabricante, y eso es funcionar muy bien.

Ian Sinclair

1

Instalación del CPC464

La caja del CPC464 contiene el ordenaor en sí, una cinta de demostración y un manual. Además de esto se necesita, o bien el monitor de fósforo verde o el monitor en color, o bien la unidad que contiene la fuente de alimentación y el adaptador para TV. Si usted ha adquirido uno de los monitores, el equipo está completo; si no, necesitará además un televisor. El manual explica claramente cómo conectar los monitores y la fuente de alimentación. Si va a utilizar la fuente de alimentación, tendrá que hacer también una conexión a la entrada de antena del televisor. Más adelante explicaremos cómo se sintoniza el televisor para recibir las señales del CPC464.

Si utiliza un monitor, tenga cuidado con la clavija de seis patillas; evite forzarla cuando la introduzca en el zócalo. A menos que el monitor y el ordenador estén a la misma alura y muy próximos entre sí, observará que la clavija no hace un contacto demasiado estable. Yo puse el monitor en un estante, más alto que el ordenador, y tuve que estirar los cables hasta conseguir que la clavija no se saliera del zócalo. Si al encender el ordenador no aparece ninguna imagen en el monitor, trate de encajar mejor la clavija. No obstante, no se producirá ningún daño si esta clavija se desconecta, aun cuando en ese momento esté escribiendo o ejecutando un programa. En cambio, si se desconecta la otra clavija, la de alimentación, el contenido de la memoria se perderá, a menos que previamente lo haya grabado en la cinta. Por esta razón es muy importante saber cómo utilizar el magnetófono incorporado al ordenador; también hablaremos de ello en este capítulo. Sin embargo, cualquiera que sea la versión de la máquina que haya comprado, lo primero que tiene que hacer es conectarla a la red.

La clavija se conecta internamente según se indica en la figura 1.1. Sólo hay dos hilos, uno azul y otro marrón. El cable debe quedar firmemente sujeto con la abrazadera. El fusible debe ser de 3 amperios, no de los de 13 amperios que se suelen instalar en las clavijas de tres terminales. Si está usted habituado a preparar sus propios enchufes, el diagrama le será suficiente indicación. En caso contrario, haga que se lo conecte un electricista (con un fusible de 3 amperios). El electricista no necesita el ordenador; basta con que le lleve el monitor o la fuente de alimentación. Una vez preparado el enchufe, asegúrese de que la fuente de alimentación esté alejada del CPC464 y del televisor. Póngala de forma que el aire pueda circular libremente a su alrededor. Sólo se calienta moderadamente cuando está conectado el ordenador, pero en cualquier caso es preferible facilitarle la refrigeración. Si está utilizando un

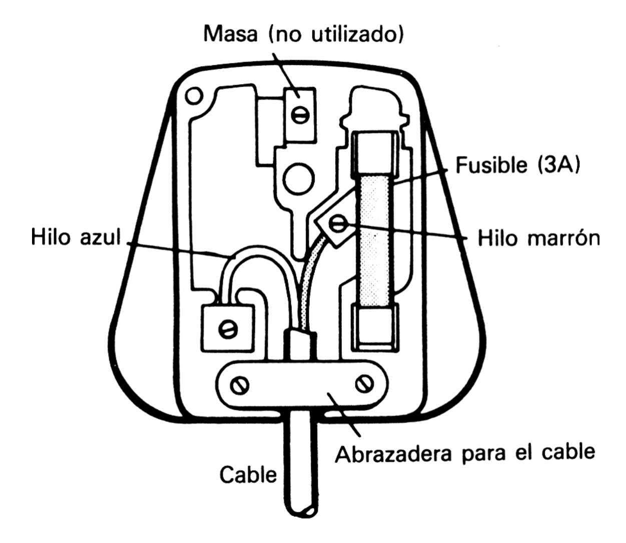


Fig. 1.1 Conexiones en la clavija de red de tres terminales. Sólo se conectan la fase y el neutro, no la masa. Si no tiene experiencia en este tipo de trabajo, es preferible que lo confíe a un electricista.

monitor, colóquelo de forma que pueda ver la pantalla completa, y a no excesiva distancia, de forma que no tenga que forzar los cables.

Si ha seguido todas estas instrucciones, ya está casi preparado para hacer funcionar el CPC464; sólo necesita, en caso de estar utilizando la fuente de alimentación, un televisor. El ordenador genera señales eléctricas con las que se forman imágenes en la pantalla. Esto se puede hacer de dos formas. Una de ellas es conectar un *monitor*, que es un dispositivo especialmente diseñado para utilizar directamente las señales procedentes del ordenador; normalmente los monitores no sirven para recibir señales de televisión. Otra forma es convertir las señales del ordenador en señales del mismo tipo que las emitidas por los transmisores de TV, de manera que puedan ser introducidas por la entrada de antena de un televisor doméstico.

Hay una diferencia muy importante entre estos dos tipos de señales. Aunque pueda ser cómodo (y más barato) utilizar un televisor, la imagen que se obtiene es de baja calidad. Esto ocurre porque los receptores de televisión no han sido diseñados para manejar señales de ordenador y por la necesidad de convertir éstas en señales compatibles con el televisor. El resultado es que las letras y otros caracteres parecen borrosos, y los colores como formados por rayas. Si usted dispone de un monitor Amstrad, que tiene un precio bastante aceptable en comparación con otros, apreciará que los signos que aparecen en la pantalla son mucho más nítidos y que los colores son mejores. El único problema que he tenido con el monitor de color Amstrad fue el control de brillo, que no daba brillo suficiente para utilizarlo a la luz del día. Si le ocurre a usted lo mismo, o si el brillo es excesivo, envíe el monitor al servicio técnico para que lo ajusten adecuadamente. No lo intente usted, a menos que sea muy experto en la reparación de televisores.

VER PARA CREER

La única forma de saber qué está haciendo el ordenador en cada momento es tenerlo conectado a un monitor o a un televisor. Para el funcionamiento del ordenador esto es indiferente, pero el usuario necesita ver en la pantalla qué está ocurriendo. Si utiliza un televisor, tendrá que conectar la salida de TV de la fuente de alimentación a la entrada de antena del televisor. A menos que pueda dedicar un televisor exclusivamente al ordenador, pronto se verá enchufando y desenchufando el cable de antena con demasiada frecuencia. Esto no es ni agradable ni conveniente, pues se corre el riesgo de deteriorar los contactos del zócalo del televisor. Una posibilidad que recomendamos es utilizar un adaptador en «Y», como el que se ilustra en la figura 1.2. Este adaptador permite tener conectados al televisor permanentemente el cable de antena y el del ordenador. Lo puede aquirir en las tiendas de componentes electrónicos y en las de televisores.

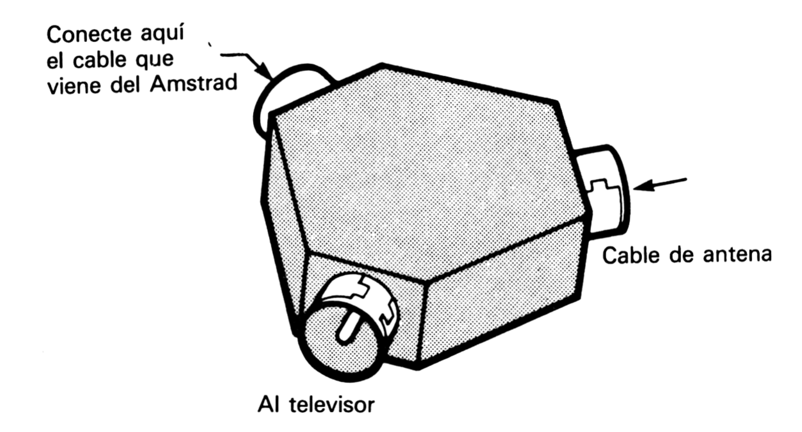


Fig. 1.2 Adaptador en «Y» para la entrada de antena del televisor; permite tener conectados simultáneamente el ordenador y la antena.

El CPC464 se conecta al televisor o al adaptador mediante el cable suministrado con la fuente de alimentación. Si este cable le resulta demasiado corto, puede poner un prolongador. Si ha instalado el adaptador en «Y», todo lo que tiene que hacer para cambiar del ordenador a la televisión es conmutar los canales. En cambio, si está utilizando un monitor, basta con que las clavijas de imagen y de alimentación estén correctamente insertadas.

El televisor que dedique al CPC464 no tiene que ser necesariamente de color, al menos para empezar. Para aprender a programar el CPC464 no necesita ver los resultados en color mientras no lleguemos al capítulo 8. En un televisor de blanco y negro los colores aparecen como gradaciones de gris. En cambio, en un televisor de color podrá ver los colores en todo su esplendor, aunque no todos los televisores dan imágenes de la misma calidad.

ENCENDIDO DEL ORDENADOR

Antes de empezar a enchufar todo lo que tiene a la vista, compruebe de cuántas tomas de corriente dispone. Si utiliza un monitor le basta con una. Si utiliza la fuente de alimentación y un televisor, necesita dos tomas de corriente, una para el CPC464 y otra para el televisor. La mayor parte de las viviendas están escasas de enchufes; puede ser necesario que se construya o compre una regleta de tomas de corriente como la que se muestra en la figura 1.3. Aun cuando en este momento le baste con una toma mural, lo más normal es que no esté donde la necesita. Además, el CPC464 es un ordenador tan atractivo que muy pronto querrá conectarle una impresora, y no mucho más tarde unidades de disquette. Todos estos aparatos necesitan sus respectivas tomas de corriente. La regleta le permite organizar mejor las conexiones y le ayuda a evitar el desastre que representaría tirar el ordenador al suelo si tropezara con un cable. No confíe en los «ladrones» o enchufes de tres vías, en los que los contactos nunca son seguros. El CPC464 tiene un interruptor de encendido, pero *siempre* se debe desenchufar el cable al terminar una sesión de trabajo. El interruptor que hay en el monitor también corta el suministro de corriente al ordenador, puesto que el monitor contiene la fuente de alimentación.

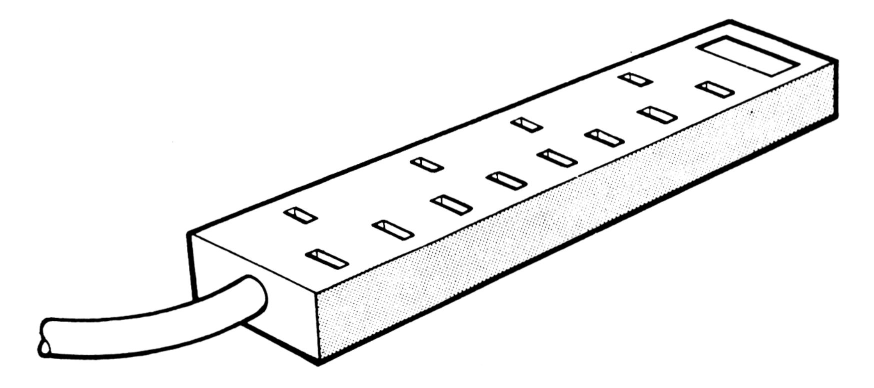


Fig. 1.3 Regleta de cuatro tomas, preferible a los «ladrones» o enchufes múltiples.

Aunque empiece con el equipo mínimo, consistente en el teclado, la fuente de alimentación y un televisor (o bien un monitor), necesitará una superficie de trabajo bastante amplia. Más adelante puede necesitar una impresora, unidades de disquette u otros periféricos (con lo que dispondrá de un *sistema* informático, no simplemente de un ordenador). Para todos estos aparatos hace falta espacio. Le podemos recomendar que adquiera un mueble de diseño especial, como el que se ilustra en la figura 1.4. Si no quiere llegar a tanto, una mesa suficientemente grande puede servirle por ahora. La informática es como la alta fidelidad: siempre hay algo nuevo que comprar.

Ahora que ya lo tiene todo instalado y conectado, le haremos algunas advertencias. El CPC464, como casi todos los ordenadores domésticos, utiliza un magnetófono de cassettes para almacenar la información. Esto se debe a que la in-

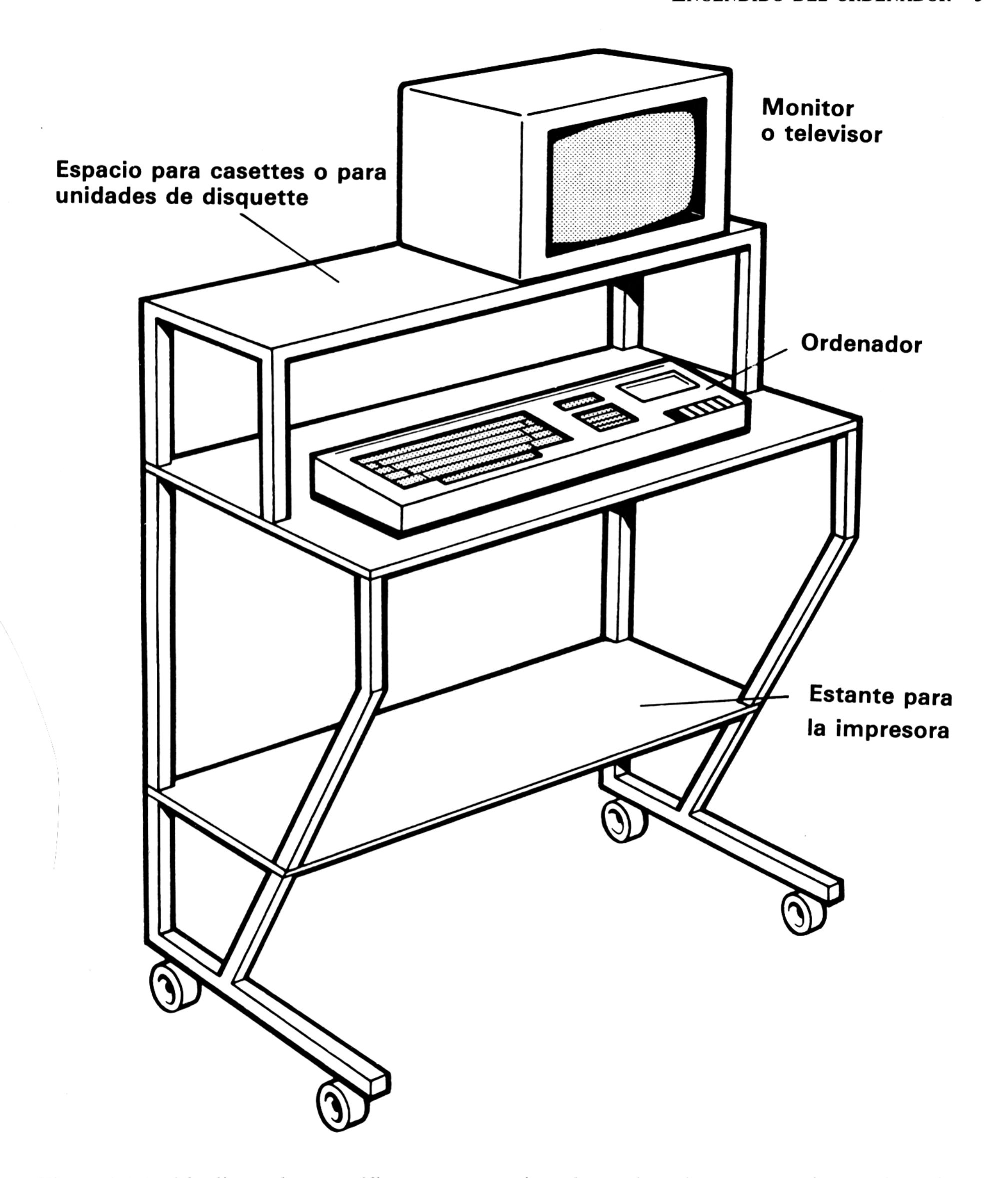


Fig. 1.4 Mueble diseñado específicamente para instalar ordenadamente un sistema basado en el CPC464.

formación que el ordenador almacena en su propia memoria se pierde en cuanto se lo desconecta. El magnetófono puede grabar señales en una cinta, y éstas quedan a salvo aunque se desconecte el ordenador. Cuando se reproducen las cintas, la información vuelve a ocupar la memoria del ordenador y éste puede utilizarla. A diferencia de la mayor parte de los ordenadores de su clase, el CPC464 tiene incorporado un magnetófono. Esto tiene varias ventajas: por ejemplo, hace innecesarios

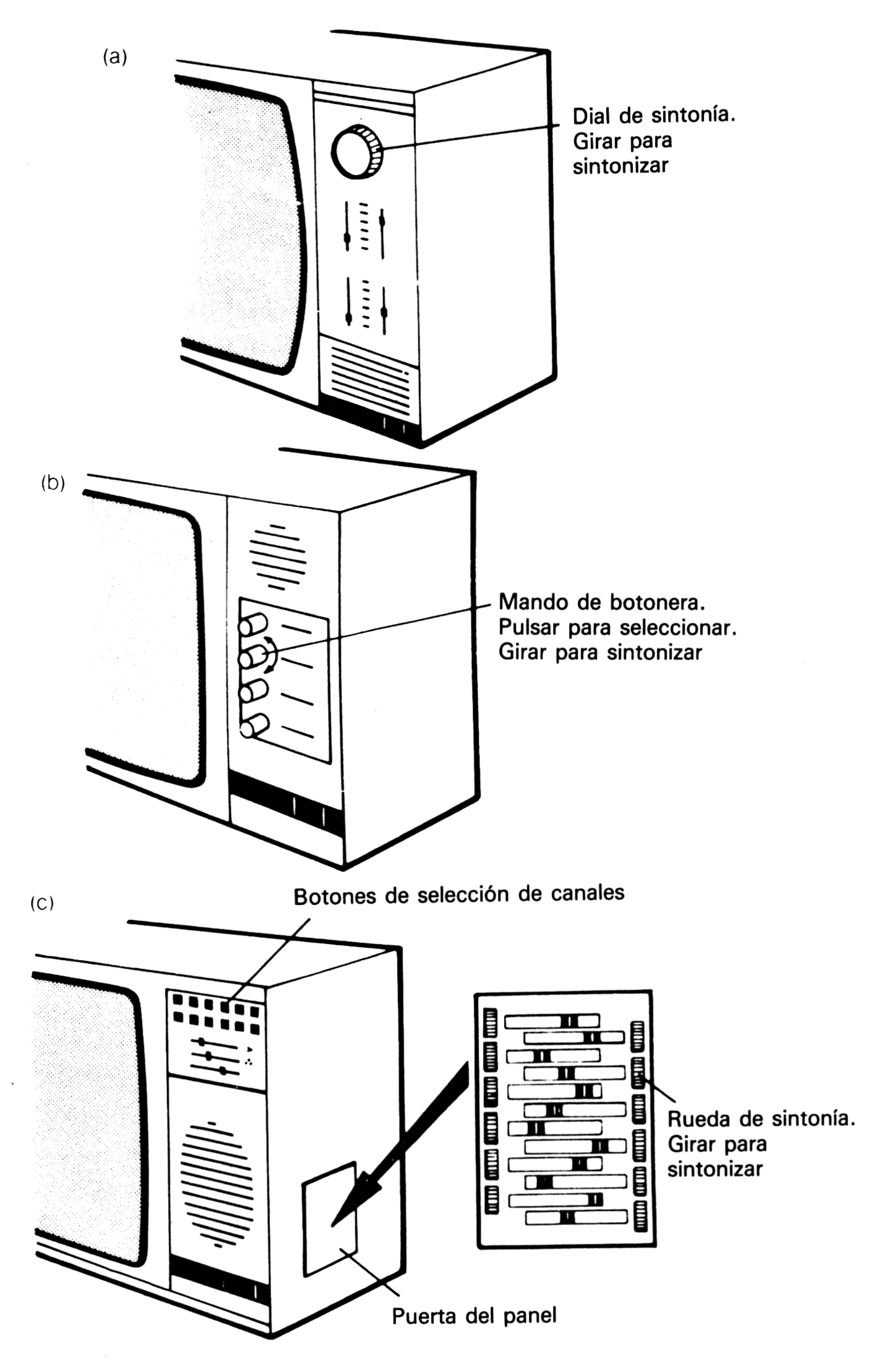


Fig. 1.5 Sintonización de televisores. (*a*) Dial de los televisores portátiles. (*b*) Mando de botonera. (*c*) Botones de contacto de los televisores más modernos.

los delicados ajustes que de otra forma habría que efectuar en un magnetófono independiente y elimina un par de cables adicionales. Por otra parte, el magnetófono incorporado ha sido diseñado para manejar señales de ordenador, y en consecuencia es mucho más fiable como medio de almacenamiento de información que los de uso general. Pero la cinta no es perfecta, ni mucho menos; vale la pena hacer dos copias de los programas especialmente valiosos. (La segunda copia es lo que se denomina copia de seguridad o *back-up*.)

Nuestra siguiente etapa va a consistir en encender el televisor (o el monitor) y el CPC464. Si lo que utiliza es un televisor, lo más normal es que en principio no vea absolutamente nada en la pantalla. La razón es que el televisor tiene que ser sintonizado para recibir las señales del ordenador. Vamos a ver, pues, cómo se realiza esta sintonía. Si tiene conectado un monitor, no es necesario que lea esta sección.

En la figura 1.5 se muestran los tres métodos más frecuentes de sintonía de televisores. El más sencillo es el de la figura 1.5*a*. Éste es el tipo que se encuentra en los televisores portátiles de blanco y negro. Para visualizar las señales del CPC464 basta con girar el mando de sintonía. Si éste está marcado con números, pruebe en la zona comprendida entre el 30 y el 40. Si no lo está, gire el mando completamente en un sentido, y luego gírelo lentamente en el sentido contrario hasta que aparezca la imagen en la pantalla.

Lo primero que va a ver, en el supuesto de que no haya tocado el CPC464 desde que lo encendió, es el texto que se muestra en la página F1.6 del manual. La segunda línea de este texto es un mensaje de copyright. Cuando aparezca en la pantalla este mensaje, gire el mando muy despacio, a un lado y a otro, hasta que consiga la máxima nitidez posible en las letras. En un televisor de color las letras no van a ser perfectas, pero procure al menos que sean estables y lo más nítidas posible.

Los televisores más antiguos de blanco y negro tienen botones de sintonía mecánicos (Fig. 1.5*b*). Normalmente son cuatro botones; de ellos hay que dedicar uno, quizá el último, a sintonizar el ordenador. Presione firmemente el botón hasta que encaje y gírelo para sintonizar. Empiece por girarlo completamente en sentido contrario al de las agujas del reloj; no le sorprenda si tiene que darle muchas vueltas antes de que llegue al tope. Ahora gire el botón en sentido contrario hasta que aparezca la imagen. Si no lo consigue en ninguna posición, compruebe si el cable está bien conectado a la entrada de antena. Si ha instalado el adaptador en «Y», compruebe si con otro botón recibe normalmente las señales de televisión. Si así es, al televisor no le ocurre nada: intente otra vez sintonizarlo con el CPC464.

Los televisores de color modernos tienen como selector de canales un conjunto de botones de presión muy pequeños, o bien botones de contacto. Estos botones sirven solamente para la selección de canales, no para sintonizar. La sintonía se realiza mediante unas ruedecitas que se encuentran en un panel, generalmente oculto tras una puerta en un lateral o en la cara frontal del televisor (Fig. 1.5*c*). Los botones suelen estar numerados, y las ruedas están marcadas con el mismo número. Elija la de número más alto (6, 8 o 12) y pulse el botón correspondiente a ese número. Para sintonizar se gira la rueda correspondiente. Al igual que en los demás casos, su objetivo es lograr una imagen lo más nítida posible y silencio en el altavoz. Los televisores de este tipo suelen realizar automáticamente una sintonía fina cuando se cierra la puerta del panel; por lo tanto, ciérrela, pues de lo contrario los circuitos

que se encargan de mantener la sintonía no funcionan y usted tendrá que estar continuamente resintonizando. El CPC464 es capaz de dar una imagen de buena calidad en casi todos los televisores. Si en el suyo la imagen es inestable o los colores son borrosos, compruebe la sintonía. Si el problema persiste, haga que ajusten su televisor, o cambie de modelo.

Cuando se utiliza un monitor la imagen suele ser buena desde el principio, prácticamente sin necesidad de ajustes. Lo que sí habrá que ajustar es el mando de brillo para adaptarlo a la iluminación del lugar de trabajo. El monitor de color tiene solamente un mando de sincronización vertical accesible (el de sincronización horizontal se puede accionar con un destornillador), pero es muy improbable que estos mandos requieran ajuste; inténtelo solamente si la imagen es muy inestable.

MANOS A LA OBRA

Ahora que ya tenemos en la pantalla la imagen enviada por el CPC464, ha llegado la hora de empezar a conocer el ordenador. Inmediatamente por debajo de la palabra «Ready» puede ver un cuadrado brillante (dorado): es el *cursor*. Sirve para indicar el lugar de la pantalla en el que aparecerá la letra o signo correspondiente a la siguiente tecla que usted pulse. Cuando lo haga, el cursor se desplazará un lugar a la derecha. Es importante advertir que nada que el usuario teclee puede ser perjudicial para la máquina; lo peor que puede ocurrir es que pierda un programa que esté almacenado en la memoria, pero nada más. Sí es posible, en cambio, dañar el CPC464 dejando que le caigan líquidos encima (¡cuidado con el café!) o conectándolo a otros aparatos cuando está encendido. También se pueden perder las señales grabadas en cinta si se extrae ésta cuando el magnetófono está funcionando, o si enciende y apaga el ordenador teniendo una cinta en el magnetófono. Apague *siempre* el ordenador y todos los aparatos a él conectados antes de conectar o desconectar alguna de las clavijas que van a la parte posterior del CPC464. No encienda ni apague *nunca* el ordenador sin previamente extraer la cinta que pudiera haber en el magnetófono; no extraiga nunca la cinta si el magnetófono está funcionando. Guarde las cintas en un lugar fresco y seco, a buena distancia del monitor (o de la fuente de alimentación).

EL TECLADO

Es hora de echar un vistazo al teclado, porque a través de él es como vamos a comunicarnos con el CPC464. Si dejamos aparte los dos grupos de teclas que están a la derecha del teclado principal, observamos que las teclas del CPC464 son muy parecidas a las de una máquina de escribir; la colocación de las letras y los números es igual. Si usted ha utilizado máquinas de escribir, en particular una electrónica, no le será difícil habituarse a este teclado. Cada vez que pulse una de las teclas literales (teclas marcadas con una letra), verá cómo aparece en la pantalla la letra correspondiente. La letra será *minúscula*, no *mayúscula*. Lo mismo que la máquina de escribir, el teclado del CPC464 normalmente produce letras minúsculas. Si lo que quiere

es escribir mayúsculas, pulse la tecla SHIFT al mismo tiempo que la de la letra deseada.

Las órdenes se le pueden dar al ordenador indistintamente en mayúsculas o en minúsculas. Si quiere escribir en mayúsculas permanentemente, pulse la tecla CAPS LOCK. Para volver a minúsculas basta con volver a pulsar CAPS LOCK. Lamentablemente, no hay una luz que indique en qué situación se encuentra CAPS LOCK; hay que probar tocando alguna tecla. Cualquiera que sea la situación de CAPS LOCK, el funcionamiento de las teclas que están marcadas con dos signos no se altera. Al pulsar una de estas teclas sin SHIFT se obtiene el signo que está grabado en la parte inferior de la tecla. Para escribir el signo superior tendrá que usar SHIFT. Por ejemplo, si pulsa la tecla del 8 sola obtendrá el número 8; si la pulsa acompañada de SHIFT obtendrá (.

Además de las teclas normales de máquina de escribir, hay otras teclas especiales. Por ejemplo, en el extremo superior izquierdo del teclado está la tecla marcada con ESC (escape); a la derecha de la barra espaciadora está la tecla CTRL (control). Estas dos teclas se utilizan de la forma que se describe en el manual. La tecla ESC es especialmente útil porque permite detener la ejecución de un programa en cualquier momento. Pulsando cualquier otra tecla se reanuda la ejecución; pulsando por segunda vez la tecla ESC se abandona definitivamente el programa. Aun cuando se haya detenido un programa de esta forma, se puede reanudar su ejecución, porque el programa continúa almacenado en la memoria. Otra forma de detener un programa es pulsar simultáneamente las teclas CTRL, SHIFT y ESC, pero entonces el programa se borra de la memoria. Utilizaremos más adelante este método, porque a veces es la forma más fácil de hacer volver el ordenador a su estado inicial. Desde luego, antes de recurrir a tan drástica medida hay que estar seguro de que se ha grabado el programa en cinta. La tecla CTRL sirve también para enviar a la pantalla signos muy diferentes de los que están marcados en las teclas; volveremos a hablar de esta cuestión. Lo que sí conviene aprender ahora es cómo borrar letras. Escriba una palabra cualquiera y pulse la tecla DEL, que está en la fila superior, a la derecha. Verá cómo desaparece la última letra tecleada. Si la mantiene pulsada, las demás letras también empezarán a desaparecer con rapidez, y el cursor se irá moviendo hacia la izquierda. Cuando no quedan más letras que borrar el ordenador le avisa haciendo sonar el altavoz interno. Si no oye el pitido, suba el control de volumen (mando que está junto al interruptor, en la cara derecha de la carcasa). La acción de la tecla DEL es una de las formas de corregir los errores mecanográficos.

Las cuatro teclas marcadas con flechas son las que controlan el movimiento del cursor. Compruebe cómo al pulsarlas el cursor se mueve por la pantalla en la dirección indicada por las flechas. Estas teclas se utilizan, junto con la COPY, en uno de los métodos de «edición» (corrección de texto e instrucciones). Este método se explica detalladamente en el apéndice A; léalo cuando crea que está preparado para ello. Finalmente, el grupo de doce teclas que está situado a la derecha del teclado principal es lo que se denomina teclado numérico. Está pensado para facilitar la introducción de números (para los diestros, naturalmente; para los que somos zurdos es inútil.)

La tecla más importante de todas, al menos para nosotros en este momento, es la tecla ENTER. Está en la posición equivalente a la de retorno del carro en las má-

quinas de escribir electrónicas, pero su acción no es la misma. Al pulsar ENTER se está diciendo al ordenador que se ha terminado de escribir una instrucción o una orden. Si está acostumbrado a manejar una máquina de escribir tendrá que cambiar alguno de sus hábitos. Por ejemplo, en una máquina de escribir tiene que pulsar la tecla de retorno del carro cada vez que llega al final de una línea, para luego continuar escribiendo al principio de la siguiente. La tecla ENTER del ordenador hace bastante más que esto. Si el texto que está introduciendo es más largo que la longitud de una línea de la pantalla, el ordenador se encarga de pasar *automáticamente* a la línea siguiente. La tecla ENTER no se debe utilizar con este propósito, sino solamente cuando se desea que el ordenador almacene una instrucción o ejecute una orden. No obstante, siempre que pulse ENTER el cursor saltará al principio de la línea siguiente. La posición en la que va a aparecer la próxima letra o signo es siempre la indicada por el cursor.

Observe que la acción de las teclas se repite, y muy deprisa, si las mantiene pulsadas durante algún tiempo. Si escribe algo que el ordenador no entienda y luego pulsa ENTER, la máquina responderá con un mensaje de error:

Syntax error

y en la línea siguiente aparecerán la palabra «Ready» y luego el cursor. El error de sintaxis se produce porque el ordenador es una máquina bastante tonta. Sólo entiende unas cuantas palabras, las denominadas *palabras reservadas*. Si lo que se le comunica no está entre estas palabras reservadas, o si se las utiliza incorrectamente, para el ordenador se ha cometido un error de sintaxis. Puede tener sentido para las personas, pero no para la máquina. La palabra «sintaxis» se refiere a la forma en que se utilizan las palabras de un idioma, y lo que el ordenador entiende es una especie de idioma.

COMPROBACIÓN DEL MAGNETÓFONO

El ordenador CPC464, como los demás, almacena las instrucciones e información de todo tipo en su memoria, pero sólo mientras está encendido. En cuanto lo apague, el contenido de la memoria desaparecerá, y de forma irrecuperable, aun cuando vuelva a encenderlo inmediatamente. Por esta razón es necesario guardar los programas y toda la información que éstos utilicen en algún otro medio de archivo. Los soportes más utilizados son los discos y las cintas magnéticas. Las unidades de disco para este ordenador son opcionales. Los discos tienen la ventaja sobre las cintas magnéticas de ser mucho más fiables y rápidos.

El ordenador dispone de circuitos que convierten las instrucciones de un programa en señales eléctricas aptas para ser grabadas en cassette mediante el magnetófono incorporado. Cuando se reproducen estas cintas, otros circuitos se encargan de convertir las señales leídas en instrucciones de programa. De esta forma, el CPC464 permite grabar los programas en cinta y su lectura posterior. Antes de abordar el resto del libro, es conveniente que se familiarice con el uso del magnetófono para grabar y leer programas.

Lo más fácil para empezar es leer (o *cargar*) un programa. Hay gran variedad de

programas disponibles para el CPC464. Como se puede leer en las notas que acompañan los programas, algunos de ellos no son fáciles de copiar porque han sido «protegidos» adrede. Normalmente no es aconsejable comprar programas que no puedan ser copiados, porque de ellos no podrá hacer copias de seguridad. Es esencial tener copias de seguridad de los programas más valiosos, y no porque la cinta no sea fiable, sino porque las copias de seguridad son una salvaguarda rutinaria en informática. Todos los sistemas de grabación están expuestos a la pérdida de señales; los profesionales de la informática guardan varias copias de todo lo que consideran importante. Así pues, parece absurdo invertir dinero en la adquisición de un programa que no se pueda copiar. Si el fabricante del programa anuncia orgulloso que su programa no puede ser copiado, sencillamente *no lo compre*. En cualquier caso, a poca experiencia que tenga en el uso de cassettes, siempre puede hacer copias por audio, utilizando dos magnetófonos y el cable adecuado.

Cargue la cinta de demostración («Welcome») que viene con su ordenador, siguiendo atentamente las instrucciones del manual. Cuando oiga que se para la cinta, pulse la tecla STOP del magnetófono. Esto es importante. Aunque el ordenador se encarga de parar el motor, la cinta continúa oprimida entre el cilindro metálico y la rueda de caucho que constituyen el mecanismo de arrastre. Si la rueda de caucho permanece ejerciendo presión contra el cilindro metálico durante algún tiempo, su borde externo se aplanará en el punto de contacto, y esta irregularidad puede causar problemas más tarde. Si esto llega a ocurrir, la reparación no será tan sencilla como en un magnetófono independiente. Tendrá que llevar el ordenador entero al taller, ya que el magnetófono está incorporado y no puede ser extraído fácilmente. Cuando pulse la tecla STOP el ordenador comenzará la ejecución del programa. Solamente algunas cintas suministradas por el fabricante del ordenador y por fabricantes de programas deben ser cargadas de esta manera. Ahora debe aprender cómo grabar sus propios programas.

GRABACIÓN

Antes de nada, necesita tener algún programa que grabar, y para ello tendrá que escribirlo. Es conveniente, para partir de cero, que apague y vuelva a encender el ordenador, salvo que acabe de hacerlo.

Escriba el número 10 y la palabra «rem». Es indiferente que escriba rem o REM. Ésta es una palabra reservada y, comoquiera que la escriba, el ordenador la convertirá (más tarde) a mayúsculas. Compruebe que no se ha equivocado y pulse ENTER. El efecto de esta acción es pasar la línea de instrucción «10 REM» a la memoria del CPC464. Cuando escribió el primero dígito, el 1 apareció en la pantalla en la posición del cursor. Al pulsar ENTER el cursor ha saltado a la línea siguiente, pero la instrucción continúa estando en el mismo lugar de la pantalla. Si la escribió en minúsculas, seguirá estando en minúsculas:

```
10 rem
```

Si observa que se equivoca al escribir, corrija utilizando la tecla DEL, con lo que el cursor se va desplazando hacia la izquierda a la vez que borra los caracteres. Una

vez borrada la letra incorrecta, escriba la correcta. Cuando haya corregido la línea entera, pulse ENTER. Si después de todo se ha equivocado y ha escrito algo como REN o RWM, el ordenador no le avisará; en realidad, el ordenador acepta cualquier cosa que se escriba tras un número. Para corregir una línea tal como

20 Ren

puede, en lugar de utilizar DEL, escribir la versión correcta:

20 rem

y luego pulsar ENTER. Escriba ahora todas las restantes líneas de la figura 1.6; no olvide pulsar ENTER al terminar cada línea. Los números se denominan *números de línea* y tienen dos cometidos. Primero, indican al ordenador que lo que les sigue es una línea de programa; segundo, sirven como guía para el ordenador, pues éste normalmente ejecuta las instrucciones en el orden de los números de línea. En todo momento puede comprobar cómo va quedando su programa sin más que pedir al ordenador que lo «liste». En informática, *listar* significa hacer que el ordenador escriba una lista con el contenido de parte de su memoria. Puesto que ha escrito los números de línea, puede estar seguro de que el ordenador ha ido guardando las instrucciones en su memoria; si escribe «list» y luego pulsa ENTER, verá el programa completo en la pantalla. No le sorprenda ver que el ordenador ha convertido letras minúsculas en mayúsculas (por ejemplo, rem en REM); otra cosa que habrá hecho es ordenar las instrucciones por número de línea y dejar un espacio entre el último dígito del número y la primera letra de la instrucción. Compruebe mediante el «listado» que su programa es idéntico al de la figura 1.6.

```
10 REM
20 REM
30 REM
40 REM
```

Fig. 1.6 Programa para comprobar las funciones de grabación y lectura del cassette.

Ahora prepare una cinta. Elija una nueva, de las especiales para ordenador, preferiblemente de duración C12 o C15. Estas cintas se encuentran en todas las tiendas especializadas en microordenadores. Introduzca la cinta en el magnetófono (según describe claramente el manual) y pulse la tecla REW para rebobinarla completamente. Ponga a cero el contador del magnetófono y pulse la tecla FF hasta que el contador señale 002. Esto es *muy importante*; al principio de todas las cintas hay un tramo de plástico (guía de cinta) no recubierto de material magnético y por lo tanto no grabable. Este tramo debe ser bobinado, de forma que frente a las cabezas quede material sensible. Por otra parte, es posible que el principio de la parte magnética de la cinta esté algo rayado. Asegúrese de que ha avanzado la cinta lo suficiente.

Veamos ahora cómo hacer una grabación. Escriba lo siguiente:

SAVE "PRUEBA"

y pulse la tecla ENTER. La palabra «SAVE» es la instrucción que indica al ordenador que queremos almacenar (es decir, grabar) un programa en cinta. La palabra «PRUEBA» es el *nombre del fichero* por el que el ordenador reconocerá el programa cuando se le pida que lo lea. Es posible grabar un programa sin darle nombre. Por ello basta con escribir SAVE y abrir comillas (''), pero esto no es una buena costumbre. Es frecuente grabar varios programas en una misma cinta, y los nombres de los programas son necesarios para ordenar a la máquina que lea el que se desea cargar. Después de pulsar ENTER aparece en la pantalla el siguiente mensaje:

Press REC and PLAY then any key

(pulse REC y PLAY, y luego cualquier tecla). Efectivamente, debe pulsar firmemente las teclas REC y PLAY del magnetófono; en cuanto a la parte «cualquier tecla» del mensaje, esto no es exactamente así: las teclas SHIFT, CAPS LOCK y CTRL no tienen ningún efecto, y la tecla BREAK interrumpe el proceso. Es decir, debe pulsar cualquier otra tecla. En cuanto lo haga, el motor del magnetófono empezará a moverse y poco después aparecerá en la pantalla otro mensaje:

Saving PRUEBA block 1

(grabando PRUEBA bloque 1). Si ha subido lo suficiente el control de volumen, podrá oir el molesto ruido que producen las señales en que se ha convertido su programa. Cuando termina la grabación, el motor se para y en la pantalla aparece la palabra «Ready» seguida del cursor. El programa ha quedado grabado en la cinta; el número que sigue a la palabra «block» indica el tamaño del programa. Los programas cortos, como los que veremos en este libro, pueden requerir un solo bloque. El motor del magnetófono se para automáticamente al terminar la grabación, pero recuerde que debe pulsar la tecla STOP del magnetófono para liberar la cinta.

Vamos a comprobar que la grabación ha sido correcta. Escriba NEW (nuevo) y pulse ENTER, con lo que borrará el programa de la memoria. Para evitar confusiones, borre la pantalla escribiendo CLS y pulsando ENTER. Ahora escriba LIST y pulse ENTER; acaba de pedir al ordenador que escriba en la pantalla el listado de las instrucciones del programa actual y, lógicamente, no ha escrito nada. Otra forma de hacer todo esto es pulsar simultáneamente las teclas CTRL, SHIFT y ESC para borrar la memoria (es decir, para simular que acaba de encender el ordenador).

Ahora puede leer el programa de la cinta. Escriba LOAD ''PRUEBA'' y pulse ENTER. El ordenador le responde con otro mensaje; esta vez le pide que pulse PLAY en el magnetófono (pero *no* REC) y luego «cualquier tecla». Hecho esto, el motor empieza a mover la cinta y en la pantalla aparece el mensaje:

Loading PRUEBA block 1

(cargando PRUEBA bloque 1). Si el volumen está suficientemente alto podrá oir los ruidos de antes. Cuando termina la lectura del programa, el motor se para automáticamente y usted debe pulsar la tecla STOP del magnetófono. Escriba LIST y pulse ENTER. Si todo ha ido bien en la grabación y en la lectura, en la pantalla aparecerán las instrucciones de su programa. Cuando haya adquirido soltura suficiente para grabar y cargar programas, estará en condiciones de empezar a programar. Como comprobará en el futuro, si ha dedicado una hora a introducir un programa por el

teclado, vale la pena dedicar unos minutos más a grabarlo en cinta para ya no tener que volver a escribirlo. El magnetófono del CPC464 es muy fiable, si bien el tiempo que tarda en grabar y reproducir es mayor que en otros ordenadores. De lo único que tiene que preocuparse es de conservar correctamente sus cintas. En la tabla de la figura 1.7 se dan algunas instrucciones a este respecto.

1. Guarde las cintas en un lugar fresco y seco, a salvo de condensaciones de vapor de agua.
2. Manténgalas alejadas de los imanes; en particular, de altavoces, televisores y motores eléctricos.
3. No toque nunca la superficie de la cinta con los dedos.
4. Si se enreda la cinta, golpéela suavemente contra la palma de la mano y trate de bobinarla en ambos sentidos en modo rápido (con las teclas FF y REW).
5. No utilice nunca el principio de la cinta.
6. No tenga las cintas almacenadas durante mucho tiempo sin bobinarlas y rebobinarlas, aunque sea en modo rápido.

Fig. 1.7 Lista de precauciones que se deben tomar para la conservación de las cintas magnéticas.

Más adelante, cuando trabaje con programas mucho más largos, la lentitud del magnetófono empezará a parecerle irritante. La solución óptima es utilizar discos. Pero aun con cinta se puede mejorar la velocidad. En efecto, si escribe

SPEED WRITE 1 (y luego pulsa ENTER)

la velocidad de grabación se duplica. La cinta sigue moviéndose a la misma velocidad, pero la transmisión de señales es más rápida. No es necesario dar esta orden al cargar los programas, pues el ordenador elige automáticamente la velocidad de lectura correcta. Desde luego, la fiabilidad no es tan grande, pero si las cintas son de buena calidad y están bien conservadas, no notará la diferencia. Lo que sí es apreciable es la diferencia de velocidad cuando se manejan programas largos. Es una buena idea conservar una copia de seguridad grabada a baja velocidad y utilizar normalmente una copia de trabajo grabada a velocidad alta. Para que el ordenador vuelva a velocidad normal (baja), basta con escribir:

SPEED WRITE 0 (y luego pulsar ENTER)

o bien reinicializar la máquina pulsando simultáneamente CTRL, SHIFT y ESC (pero recuerde que de esta forma se borra el programa).

CARGA Y EJECUCIÓN INMEDIATA DE PROGRAMAS

Más adelante, cuando esté escribiendo sus propios programas, puede desear cargarlos y ejecutarlos inmediatamente, lo mismo que hacíamos con la cinta de demostración. El método es muy sencillo. Escriba la orden SAVE seguida del nombre del pro-

grama tal como hemos hecho antes; escriba a continuación la letra P (de «protegido»). Un ejemplo sería:

SAVE "proglargo",P

El proceso de grabación es como antes, pero lo que queda grabado es distinto. Para leer este programa hay que hacer una de las dos cosas siguientes:

1. Pulsar CTRL y la tecla ENTER *pequeña*, luego pulsar PLAY en el magnetófono y «cualquier tecla» en el teclado.
2. Escribir RUN", pulsar la tecla ENTER grande y luego actuar con el magnetófono como siempre.

En cualquier caso, el ordenador cargará y ejecutará el programa. Sin embargo, una vez terminada la ejecución, el programa ya no puede ser listado ni ejecutado. No grabe nunca un programa protegido de esta forma a menos que disponga de otra copia no protegida; si surge algún problema el programa se habrá perdido y ya no podrá copiarlo. Los métodos de protección ayudan mínimamente a reducir la piratería de programas, pero castigan indebidamente al usuario honrado que no sabe cómo hacer copias de seguridad de los programas protegidos. Cuando haya adquirido alguna experiencia en programación empezará a encontrar en las revistas sugerencias sobre cómo burlar estos métodos de protección.

En una primera lectura del libro no es necesario que lea el resto de esta sección, pues su contenido le será útil solamente cuando haya adquirido cierta experiencia. Además de LOAR y RUN, hay otras dos formas de cargar un programa en memoria. Una de ellas es teclear la palabra CHAIN seguida del nombre del programa entre comillas. El efecto es cargar y ejecutar el programa, de modo que CHAIN sustituye a LOAD y RUN. Otra forma es utilizar MERGE. Normalmente, cuando se carga un programa se borra el actualmente presente en la memoria; MERGE, en cambio, *mezcla* el programa nuevo con el actual. Los números de línea deben ser distintos pues, de lo contrario, las instrucciones del programa nuevo sustituyen a las de igual número del antiguo. Se puede combinar CHAIN con MERGE (por ejemplo, CHAIN MERGE "PROGR") para mezclar un programa nuevo con el actual y ejecutar el resultante. Por otra parte, se puede hacer una lista del contenido de una cinta mediante la orden CAT.

2

Manejo de la pantalla

Después de leer el capítulo 1 el lector ya se habrá percatado de que el CPC464, como casi todos los demás ordenadores, recibe las órdenes del usuario a través del teclado. También habrá observado que el ordenador tiene en cuenta una orden cuando se pulsa ENTER, no antes. Ya hemos utilizado la orden LIST, que hace escribir en la pantalla la lista de las instrucciones de que consta el programa. Hay dos detalles útiles que debe conocer antes de seguir adelante. Uno es que se puede borrar la pantalla escribiendo CLS (o cls) y pulsando ENTER. Cuando se haya familiarizado con el teclado querrá utilizar las diversas órdenes de edición; éstas están descritas en el apéndice A.

Hay dos formas de utilizar el ordenador. Una es el denominado *modo directo;* consiste en escribir una orden y pulsar ENTER, con lo cual la máquina obedece la orden inmediatamente. Esto puede ser útil en ocasiones, pero la forma usual de utilizar el ordenador es en *modo de programa.* En modo de programa al ordenador se le suministra un conjunto de instrucciones junto con el orden en que debe ejecutarlas. Tal conjunto de instrucciones es lo que se denomina *programa.* La diferencia es importante, porque las instrucciones de un programa se pueden repetir cuantas veces se desee sin requerir más que un mínimo esfuerzo por parte del usuario. En cambio, para repetir una orden directa hay que volver a escribirla completa y pulsar otra vez ENTER. El conjunto de palabras que representan las instrucciones y órdenes, junto con las normas para su correcta utilización, es lo que se denomina *lenguaje de programación.* El CPC464 está dotado de una versión actualizada del lenguaje más utilizado en programación de pequeños ordenadores: el BASIC. BASIC son las siglas de «Beginners All-purpose Symbolic Instruction Code» (Código de instrucciones simbólicas de uso general para principiantes). Este lenguaje fue desarrollado con objetivos didácticos, pero ha evolucionado hasta convertirse en uno de los más útiles. La versión de BASIC del CPC464 está enriquecida con numerosas instrucciones que no se incluían en el BASIC original.

Hay una cuestión importante acerca de todas las instrucciones de ordenador, tanto si son órdenes directas como si están incluidas en un programa: tienen que estar escritas de una forma precisa, ajustándose a las normas. Si hay un error de mecanografía, el ordenador sencillamente no podrá entenderlas. Además, se las debe utilizar de la forma correcta. Por ejemplo, si escribimos SAVE y pulsamos seguidamente ENTER, el ordenador no puede actuar porque después de la palabra SAVE falta

el signo de abrir comillas, es decir, la orden está incompleta. En algunas órdenes hay que separar las palabras con espacios; el ordenador no las entiende si faltan los espacios. En otros casos los espacios no tienen importancia. Estos detalles se aprenden por la experiencia, pues no hay una norma clara al respecto.

El CPC464, como ya sabemos, permite que escribamos las órdenes e instrucciones tanto en mayúsculas como en minúsculas, si bien convierte éstas en aquéllas cuando lista el programa en la pantalla. Esto puede causar confusión cuando se trata de leer un listado en la pantalla y las instrucciones se habían introducido en minúsculas. Así pues, a partir de ahora en este libro escribiremos las instrucciones en mayúsculas, tanto en el texto como en los listados. También sabemos que después de teclear las órdenes directas hay que pulsar ENTER, de modo que en lo sucesivo ya no necesitaremos especificarlo en todos los casos.

Examinemos la diferencia que hay entre una orden directa y una instrucción de programa. Si quiere que el ordenador ejecute, como orden directa, la suma de los números 1.6 y 3.2, debe escribir lo siguiente:

```
PRINT 1.6 + 3.2
```

(y, recordémoslo una vez más, pulsar ENTER a continuación). Hay que empezar por PRINT porque ésta es la única forma de indicarle al ordenador que lo que se quiere es que escriba la respuesta en la pantalla. No reconoce instrucciones tales como «ESCRIBE» o «¿CUANTO ES?»; solamente entiende unas cuantas palabras, las *palabras reservadas* o *palabras de instrucción*. PRINT es una de esas palabras. Observe que tiene que dejar un espacio después de la T. En cambio, no es necesario que haya espacios ni antes ni después del signo +.

Cuando se pulsa la tecla ENTER después de escribir PRINT 1.6 + 3.2, en la pantalla aparece la respuesta, 4.8. Esta respuesta no aparece en el mismo lugar en el que escribimos la orden, sino al principio (a la izquierda) de la línea siguiente, dejando un espacio en blanco. La instrucción PRINT es excepcional en el sentido de que puede ser escrita de dos formas. En efecto, se puede escribir el signo de cerrar interrogación (?) en lugar de PRINT. Además, si se escribe el signo de interrogación, no es necesario dejar espacio entre él y el primer número. Así, puede escribir

```
?1.6 + 3.2
```

y pulsar ENTER, y obtendrá la respuesta sin el temido mensaje de error de sintaxis. Cuando en un programa dado hay muchas instrucciones PRINT, se ahorra tiempo de mecanografía si se escribe ? en lugar de la palabra completa.

Una vez ejecutada una orden directa, el ordenador la da por terminada y la olvida. No ocurre lo mismo con las instrucciones de programa. Cuando se escribe un programa, las instrucciones no van siendo ejecutadas a medida que se pulsa ENTER al final de cada línea, sino que el ordenador las va archivando en su memoria, en espera de que el usuario le pida que las ejecute. Desde luego, el ordenador necesita poder distinguir entre órdenes directas e instrucciones. En BASIC esto se consigue haciendo que todas las líneas de programa empiecen con un número de línea. Además, los números de línea tienen que ser enteros positivos. Ésta es la razón por la que no se puede esperar que el ordenador entienda una instrucción tal como 5.6 + 3 =, porque considera que el 5 es un número de línea, y el resto no tiene sentido.

Empecemos a programar las operaciones aritméticas: suma, resta, multiplicación y división. Escogemos éstas por ser fáciles de manejar. Por supuesto, los ordenadores no se utilizan normalmente para este tipo de cálculos, pero es necesario saber cómo realizarlos. En la figura 2.1 se muestra un programa de cuatro líneas que escribe en la pantalla el resultado de otras tantas operaciones aritméticas.

```
10 PRINT 5.6+6.8
20 PRINT 9.2-4.7
30 PRINT 3.3*3.9
40 PRINT 7.6/1.4
```

Fig. 2.1 Un programa aritmético de cuatro líneas.

Observemos detenidamente estas instrucciones, porque hay mucho que aprender de ellas. Para empezar, los números de línea no son 1, 2, 3, 4, sino 10, 20, 30, 40. La razón es la siguiente: si después de escrito el programa se da cuenta de que ha olvidado una instrucción entre las líneas 10 y 20, puede sencillamente escribir un número entero cualquiera comprendido entre el 10 y el 20 (por ejemplo, el 11, el 15, etc.) y a continuación la instrucción olvidada. (El orden en que se introduzcan las instrucciones carece de importancia; lo que cuenta es el número de línea de cada una de ellas. El ordenador las coloca por orden numérico automáticamente.) Si los números de línea hubieran sido 1, 2, 3, 4, no habría hueco entre ellos para intercalar una nueva instrucción. De todas formas, esto no sería un problema insuperable, porque los números de línea se pueden modificar utilizando las órdenes de edición.

El siguiente detalle en que debe fijarse es la barra que atraviesa el número 0. Esto sirve para distinguirlo de la letra O. El ordenador no acepta el 0 en lugar de la O, ni viceversa. La diferente representación gráfica en la pantalla facilita al usuario la distinción entre estos dos caracteres. Por otra parte, en el teclado el cero está en una tecla diferente, también atravesado por la barra, y tiene forma distinta de la de la letra O. Practique hasta familiarizarse con los dos signos.

Otras dos cuestiones importantes. La estrella o, mejor dicho, el asterisco de la línea 30 es el símbolo que el CPC464 utiliza como signo de multiplicar. Así pues, no se puede usar como signo de multiplicar ni x ni X, pues estos caracteres son letras. El signo de división es la barra inclinada (/), no los dos puntos (:); la barra a que nos referimos es la que está en la misma tecla que el signo de interrogación, no en la tecla de al lado.

De momento, es de esperar que todo vaya quedando claro. El programa se introduce en el ordenador escribiéndolo tal como está en la figura 2.1. No hace falta dejar espacio entre el número de línea y la P de PRINT, porque el CPC464 lo pondrá automáticamente cuando liste el programa. Pero sí es necesario dejar espacio después de PRINT, aunque esto no es así en todos los ordenadores. Recuerde que no tiene que dejar ese espacio si escribe ? en lugar de PRINT. Cada vez que termine de escribir una línea de programa tiene que pulsar ENTER, antes de escribir el siguiente número de línea. El resultado final debe ser exactamente como se ve en

la figura. Una vez terminada la introducción del programa, éste se encuentra almacenado en la memoria del ordenador, en forma de un conjunto de códigos numéricos. Hay dos cosas interesantes que aprender ahora; primero, cómo comprobar que el programa efectivamente ha quedado archivado en la memoria; segundo, cómo hacer que la máquina lo ejecute.

Para el primer problema la solución está en la orden LIST. Si lo desea, puede borrar la pantalla con la orden CLS. Escriba LIST y pulse ENTER. El ordenador escribirá en la pantalla el listado de las instrucciones. Observe cómo ha insertado espacios a la derecha de los números de línea y cómo ha convertido el signo ? en la palabra PRINT. Para ejecutar el programa necesitamos otra orden: RUN. Escriba RUN y pulse ENTER, y verá cómo el ordenador ejecuta el programa. Más concretamente, lo que verá es lo siguiente:

```
12.4
4.5
12.87
5.42857143
```

Esta última línea puede darle una idea de precisión con la que el CPC464 realiza estos cálculos. Habrá observado, si listó el programa antes de ejecutarlo, que los resultados han aparecido debajo del listado. Para evitarlo, utilice la orden CLS antes de RUN.

Cuando después de la instrucción PRINT se escribe una operación aritmética tal como 2.8 * 4.4, lo que el ordenador imprime al ejecutar el programa es el *resultado* del cálculo; es decir, no imprime 2.8 * 4.4, sino el producto.

Esto es útil, pero los resultados impresos en la pantalla pueden no significar nada si se ha olvidado cuáles eran las preguntas. El CPC464 permite escribir cualquier cosa en la pantalla, exactamente como se desee, utilizando las denominadas *cadenas literales.*

```
10 PRINT"2+2="2+2
20 PRINT"2.5*3.5="2.5*3.5
30 PRINT"9.4-2.2="9.4-2.2
40 PRINT"27.6/2.2="27.6/2.2
```

Fig. 2.2 Utilización de las comillas. En éste, como en otros ejemplos, al introducir el programa se escribió el signo ?, pero en el listado aparece la palabra PRINT.

En la figura 2.2 se da un ejemplo. En cada una de las cuatro líneas, parte del texto está entre comillas. Teclee este programa y ejecútelo. Observe el resultado. Todo lo que en el programa estaba entre comillas ha aparecido exactamente como lo había escrito. Las operaciones que estaban fuera de las comillas han sido calculadas, y en la pantalla aparecen los resultados. La combinación de los dos efectos da

$$2 + 2 = 4$$

Pero esto no tiene nada de automático. Si escribe la siguiente línea

15 PRINT "2 + 2 = "5 * 1.5

y ejecuta el programa, obtendrá sencillamente

2 + 2 = 7.5

El ordenador no hace más que lo que se le manda. Sólo quien no los conozca puede temer que algún día los ordenadores lleguen a dominar el mundo.

Este es buen momento para insitir en algo que ya hemos dicho. La línea n.º 15 que añadió al programa ha sido intercalada entre la 10 y la 20. Si no lo cree, liste el programa. Cualquiera que sea el orden en que introduzca las instrucciones, el ordenador las entenderá en orden numérico ascendente. Observe también que el ordenador ha insertado un espacio entre el signo = y el primer dígito de la respuesta. Se trata de un hueco que deja para un posible signo + o −. Si queremos dar un espacio adicional, podemos hacerlo intercalando un espacio entre el signo = y el de cerrar comillas ('') en la línea de instrucción.

Con todos los conocimientos que hemos adquirido, ya estamos en condiciones de abordar otros métodos de visualización en la pantalla. La instrucción PRINT, utilizada como hasta ahora, solamente escribe en la pantalla. Para activar la impresora hay una variante que consiste en la palabra PRINT seguida del signo # y del número 8. Así, PRINT # 8 hará que lo que se escriba sea enviado a la impresora (naturalmente, *si hay una impresora conectada*). Si no hay impresora, no utilice esta instrucción; si lo hace, el ordenador parecerá bloqueado, sin responder al teclado. Para salir de esta situación puede, o bien conectar la impresora, o bien pulsar dos veces la tecla ESC. En el apéndice B se explica cómo utilizar una impresora.

```
10 PRINT "ESTE ES"
20 PRINT "EL EXCELENTE"
30 PRINT "ORDENADOR AMSTRAD CPC464"
```

Fig. 2.3 Utilización de la instrucción PRINT para escribir palabras en la pantalla.

Pruebe ahora el programa de la figura 2.3. Puede introducir las líneas en el orden que quiera, para demostrarse a sí mismo que en el listado siempre saldrán ordenadas. Cuando borre la pantalla y ejecute el programa, las palabras aparecerán en tres líneas distintas. Esto ocurre porque la instrucción PRINT no se limita a escribir lo especificado, sino que también provoca el salto al principio de la línea siguiente. Dicho sea de paso, observe también que cuando el texto llega a la última línea de la pantalla, todo el contenido de ésta se mueve hacia arriba y la primera línea desaparece. Este efecto se denomina *scrolling* («rodadura») pues, efectivamente, la pantalla parece moverse como un rodillo. Así es como la máquina se las arregla para visualizar muchas de las líneas en una pantalla en la que sólo caben 25.

No siempre es deseable que cada instrucción PRINT provoque el salto a la línea siguiente. Para evitarlo podemos utilizar signos de puntuación que actúan como

como *modificadores.* Antes de pasar al siguiente ejemplo prepárese para adquirir otro hábito: escriba NEW y pulse ENTER. Esto sirve para borrar el programa anterior; borre también la pantalla, si lo desea, con CLS.

Si no utiliza la orden NEW, cabe la posibilidad de que queden líneas del programa antiguo mezcladas con las del nuevo. Cada vez que se escribe una línea de programa, se anula la línea que hasta ese momento tenía ese mismo número, si había alguna. Ahora bien, si en el programa antiguo hay líneas con números que no se utilicen en el nuevo, esas líneas no se borrarán, y quedarán mezcladas en el nuevo programa. Por ejemplo, en el programa de la figura 2.2 habíamos intercalado la línea 15. Esta línea ha permanecido en la memoria a pesar de que después hemos introducido otras líneas con los números 10 y 20.

```
10 PRINT "ESTE ES ";
20 PRINT "EL EXCELENTE ";
30 PRINT "AMSTRAD CPC464"
```

Fig. 2.4 Efecto del punto y coma.

Pruebe el programa de la figura 2.4. Hay una diferencia fundamental entre éste y el de la figura 2.3, que comprobará cuando lo ejecute. El efecto del punto y coma que sigue al cierre de las comillas es inhibir el salto a la línea siguiente. Al ejecutar el programa verá que todas las palabras aparecen en la misma línea de la pantalla. En este caso habría sido más fácil haber incluido todo el texto en una sola línea de programa:

10 PRINT "Este es el excelente Amstrad CPC464"

pero hay ocasiones en que es necesario utilizar el punto y coma para forzar la escritura de dos cadenas literales en una misma línea. Veremos ejemplos de esto más adelante. Ahora vamos a probar un pequeño cambio. Modifique la línea 30 para que sea

30 PRINT "ordenador Amstrad CPC464"

y ejecute el programa. Esta vez el resultado será muy diferente. Las dos primeras líneas aparecen fundidas en una, y la tercera está separada. Esto ocurre porque, de haber tratado de escribir todo el texto en una línea, la última palabra habría quedado partida. El CPC464 no permite que ocurra tal cosa; ésta es una característica muy interesante, peculiar de esta máquina.

FILAS Y COLUMNAS

La visualización elegante del texto consiste en colocar adecuadamente los números y las palabras en filas y columnas de la pantalla. Vamos a dedicar algún tiempo a estas cuestiones. Para empezar, ya sabemos que la instrucción PRINT, en ausencia

de modificadores, provoca el salto a la línea siguiente, de forma que el efecto de esta instrucción en el programa de la figura 2.5 no debería sorprendernos. Sin embargo, las líneas 10 y 20 contienen una novedad: dos instrucciones en cada línea. Las instrucciones están separadas por el signo de dos puntos (:). Siempre que se desee se pueden escribir varias instrucciones en una línea de programa, aun cuando ocupen varias líneas físicas en la pantalla. Una limitación práctica es la dificultad de leer una línea con demasiadas instrucciones. En una línea «multi-instrucción» de este tipo, el CPC464 trata las instrucciones de izquierda a derecha. La instrucción CLS tampoco debe sorprendernos; borra la pantalla y hace que la escritura empiece en el extremo superior izquierdo. El efecto es similar al que ejerce cuando se la utiliza como orden directa, pero aquí es automático por estar incluida en el programa.

```
10 CLS:PRINT"ESTE ES EL CPC464"
20 PRINT:PRINT
30 PRINT "DISPUESTO A TRABAJAR PARA USTED"
```

Fig. 2.5 Utilización de la instrucción CLS para borrar la pantalla dentro de un programa. Líneas «multi-instrucción».

Otro detalle que se debe observar en el programa de la figura 2.5 es que la línea 20 hace que las dos líneas de texto queden separadas entre sí. En ella hay dos instrucciones PRINT, sin nada que escribir; lo único que tienen que hacer estas instrucciones es pasar a la línea siguiente, borrándola al mismo tiempo. Hay otras formas de conseguir lo mismo, como veremos, pero ésta es muy cómoda.

```
10 PRINT 1,2
20 PRINT 1,2,3,4
30 PRINT "UNO","DOS"
40 PRINT "UNO","DOS","TRES","CUATRO"
50 PRINT "ESTO ES MAS LARGO","DOS"
60 PRINT 1,2,3,4,5,6
```

Fig. 2.6 La coma sirve para escribir en columnas.

El programa de la figura 2.6 ilustra el manejo de las columnas. La línea 10 es una instrucción PRINT que actúa con los números 1 y 2. Cuando aparecen en la pantalla, están situados como si ésta hubiera sido dividida en columnas. El modificador que produce este efecto es la coma; la acción es completamente automática. En el teclado la coma está a la derecha de la M; si se utiliza el apóstrofe que está en la tecla del 7, el efecto no es el mismo. Estos dos signos son parecidos en el teclado, pero completamente diferentes en la pantalla. Como demuestra la línea 20, sólo hay

tres columnas en la pantalla, cada una de 14 caracteres de anchura (o menos, si lo que se escribe son números). Si se intenta escribir algo en la cuarta columna, el efecto es pasar a la primera columna de la línea siguiente. La coma sirve tanto para números como para palabras, como demuestran las líneas 30 y 40. Cuando escriba instrucciones de este tipo, recuerde que las comas tienen que quedar *fuera* de las comillas. Las comas que estén dentro de las comillas serán transcritas sin modificación, y no afectarán a la posición del texto en la pantalla. Observe que si intenta escribir en una columna un texto que no quepa en ella, parte del texto pasará a la columna siguiente. La línea 50 ilustra este efecto: la primera frase se parte en dos y ocupa las columnas primera y segunda. La palabra DOS pasa a la tercera columna. La línea 60 demuestra qué ocurre si escribe más cosas separadas por comas; los tres últimos números pasan a la línea siguiente.

La acción de las comas es semejante en casi todos los ordenadores personales, pero el CPC464 introduce un refinamiento. Escriba ZONE 6, pulse ENTER y ejecute el programa. Ahora los textos encajan mejor, como si hubiera más columnas en la pantalla. De hecho, las hay. El número que se escribe después de ZONE indica el espaciamiento entre columnas; de esta forma se puede dividir la pantalla (invisiblemente) en el número de columnas deseado. Esto es particularmente útil para el diseño de aplicaciones comerciales, en las que la tabulación es importante.

Las comas son útiles si lo que se pretende es crear columnas de igual anchura. Pero hay un método mucho más flexible, que se programa con la palabra reservada TAB a continuación de PRINT. TAB es abreviatura de «tabular», que significa «dividir en columnas». Para entender el funcionamiento de TAB, es necesario recordar que la pantalla, tal como la encontramos en el momento de encender el ordenador, está dividida en 40 columnas y 25 filas. En la figura 2.7 se representa gráficamente esta subdivisión. En horizontal, las posiciones están numeradas del 1 al 40, pero TAB(0) significa lo mismo que TAB(1): el extremo izquierdo. TAB(40) representa el extremo derecho. La palabra TAB debe ir a continuación de PRINT, y a su derecha se debe escribir un número entre paréntesis. Si se omiten los paréntesis, el resultado es que el ordenador escribe un cero, además de lo que se le ha mandado escribir; más adelante veremos por qué.

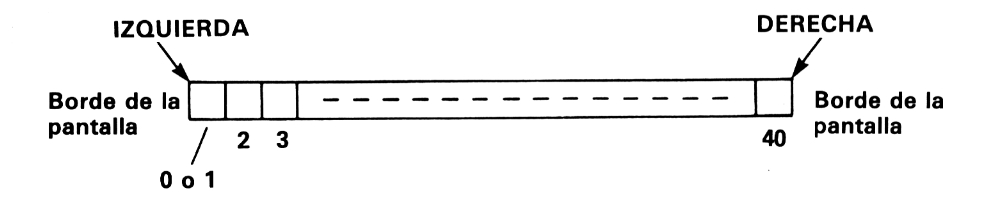


Fig. 2.7 Mapa de la instrucción TAB, que muestra los números que se deben utilizar con TAB en función de la posición deseada en la pantalla.

Probemos TAB con el ejemplo de la figura 2.8. El primer texto queda centrado en la pantalla. El segundo, cerca del borde izquierdo. El tercero, en la misma posi-

```
10 PRINT TAB(18) "CENTRO"
20 PRINT TAB(5) "EMPEZAMOS AQUI"
30 PRINT TAB(45) "ESTO ES 45"
```

Fig. 2.8 Ejemplo de cómo se utiliza TAB en un programa para colocar el cursor.

ción (en horizontal) que el segundo. Si después de TAB se escribe un número mayor que 41, el resultado es equivalente a restar 40 de ese número. Por ejemplo, TAB(45) equivale a TAB(5). Pasar de 40 no implica saltar a la línea siguiente, como ocurre en otros ordenadores. En este ejemplo la palabra «Centro» ha quedado centrada utilizando TAB(18). En la figura 2.9 se indica cómo se calcula el número que se debe utilizar para centrar una palabra o una frase. Tenga en cuenta, no obstante, que la pantalla que el CPC464 genera no está perfectamente centrada en el monitor. Por ejemplo, en mi monitor de color el borde izquierdo es más ancho que el derecho.

---

1. Cuente el número de caracteres de la palabra o frase que quiera centrar.
2. Reste este número de 40 (si es par) o de 41 (si es impar).
3. Divida el resultado por 2.
4. Utilice el número resultante con TAB.

Fig. 2.9 Fórmula para centrar textos.

En la figura 2.10 se muestra una forma diferente de espaciar números y palabras en la pantalla. En este ejemplo se utiliza la instrucción SPC, que es diferente de TAB, a pesar de las apariencias. Cuando se especifican posiciones de TAB, la primera letra de lo que se escribe a continuación se sitúa en la posición indicada. Dicho sea de paso, a continuación de PRINT se pueden poner antos TAB como se desee. A diferencia de la mayor parte de los ordenadores, el CPC464 no requiere el uso del signo punto y coma para separar los elementos de esta instrucción. Veamos el funcionamiento de SPC en la línea 20. El efecto no es tabular hasta la posición 16, sino escribir 16 espacios entre el final de la palabra ORDENADOR y el principio de CPC464. Esto es muy diferente de poner la C de CPC464 en la 16.ª posición de tabulación. La regla general es la siguiente: utilizar TAB cuando se desea tabular en columnas; utilizar SPC cuando se desea especificar la separación entre palabras o números, aunque éstos tengan anchuras diferentes.

```
10 PRINT TAB(2)"ORDENADOR"TAB(16)"CPC464"
20 PRINT TAB(2)"ORDENADOR"SPC(16)"CPC464"
```

Fig. 2.10 Utilización de SPC.

Todavía hay otra forma de especificar la posición de escritura, más flexible que las que ya conocemos. Se trata de la instrucción LOCATE, que permite situar un texto en cualquier lugar de la pantalla. Normalmente, la instrucción PRINT hace que el ordenador pase a la línea siguiente y no permite retroceder a la anterior. Utilizando LOCATE se puede escribir en cualquier sitio, e incluso reemplazar una línea con otra.

En el CPC464 la instrucción LOCATE tiene que preceder a la instrucción PRINT. Puede estar en la línea inmediatamente anterior a la de PRINT, o incluso en la misma línea, separadas por el signo de dos puntos (:).

LOCATE va seguida de dos números (parámetros). El primero es el número de columna, es decir, la posición en horizontal, entre 1 y 40. El segundo parámetro es el *número de fila* en la pantalla. Las líneas de la pantalla se numeran de 1 (la superior) a 25 (la inferior).

```
10 CLS
20 LOCATE 1,1:PRINT "ARRIBA IZQUIERDA"
30 FOR N=1 TO 1000:NEXT
40 LOCATE 28,25:PRINT "ABAJO DERECHA";
50 FOR N=1 TO 1000:NEXT
60 LOCATE 1,25:PRINT "ABAJO IZQUIERDA"
70 FOR N=1 TO 1000:NEXT
80 LOCATE 27,1:PRINT "ARRIBA DERECHA"
90 FOR N=1 TO 1000:NEXT
100 LOCATE 17,12:PRINT "CENTRO"
```

Fig. 2.11 Cómo utilizar LOCATE para escribir en cualquier lugar de la pantalla.

Es conveniente ilustrar esta instrucción con un ejemplo. Observe el programa de la figura 2.11. La línea 10 borra la pantalla. La línea 20 introduce la primera instrucción LOCATE; aquí 1,1 significa que la escritura se debe realizar en el extremo superior izquierdo de la pantalla. Es impescindible dejar un espacio entre la E de LOCATE y el primer número. La A de ARRIBA IZQUIERDA aparecerá en el rincón superior izquierdo, como en todo caso ocurriría después de CLS. La siguiente línea no es más que una pérdida de tiempo (un retardo); más adelante estudiaremos estas instrucciones.

La siguiente instrucción LOCATE va seguida de 28,25, de modo que la última A de ABAJO DERECHA queda en el extremo inferior derecho de la pantalla. ¿Cómo hemos calculado estos números? Muy sencillo: la última A de DERECHA tiene que estar en la posición 40; así vamos contando hacia atrás, 39, 38, . . . , hasta llegar a la A de ABAJO, que estará en la posición 28. Como el texto ha de estar en la última línea, el segundo parámetro tiene que ser 25. Para evitar el scrolling (desplazamiento hacia arriba) de la pantalla, hay que añadir el punto y coma al final de la línea 40. Normalmente no sería necesario poner este punto y coma; este caso es

excepcional porque se alcanza la última posición de escritura de la pantalla. Con estas explicaciones usted puede continuar el análisis del programa. Los retardos sirven para permitirle observar el orden en que el programa va escribiendo los textos.

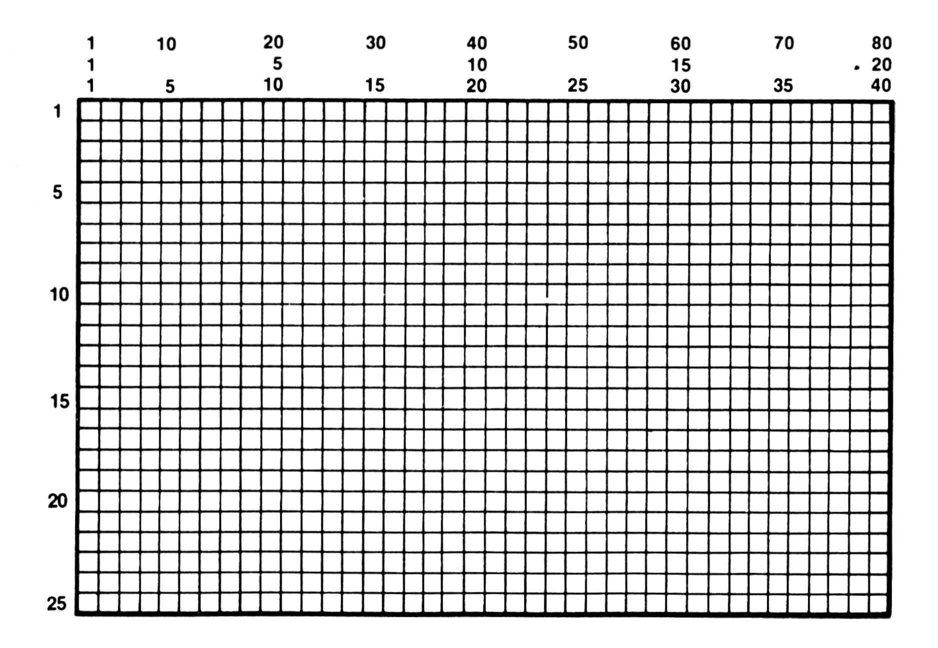


Fig. 2.12 Mapa para facilitar la utilización de la instrucción LOCATE.

En la figura 2.12 damos un mapa de la pantalla, muy útil para la colocación de textos en ella mediante la instrucción LOCATE. En este mapa se dan varios conjuntos de números; y usted se preguntará por qué. La razón es que podemos elegir tres tamaños diferentes de caracteres. No borre de la memoria el programa de la figura 2.11 y escriba MODE Ø (y a continuación pulse ENTER); no olvide dar un espacio entre la E y el 0. Si ahora hace un listado del programa (LIST y ENTER), verá que los caracteres son el doble de anchos que antes. Si ejecuta el programa, observará que los textos de la izquierda están bien colocados, pero no así los de la derecha. Esto ocurre porque ahora los parámetros de LOCATE deberían estar entre 1 y 20. Escriba ahora MODE 2 (ENTER). Los caracteres se han hecho mucho más estrechos (80 por línea). También son más difíciles de leer (salvo en el monitor de fósforo verde). Los parámetros de LOCATE van ahora de 1 a 80. Cuando se enciende el ordenador, el modo de pantalla es siempre el 1, con 40 caracteres por línea. Si no está utilizando el monitor de fósforo verde, evite el modo 2.

Para terminar, resaltaremos que el CPC464 ofrece dos funciones que muy pocos ordenadores tienen. Una de ellas es AUTO. Si escribe AUTO (ENTER), la máquina escribirá el siguiente número de línea automáticamente; el usuario sólo tiene que escribir las instrucciones que hayan de ir en cada línea. Cada vez que pulse ENTER,

aparecerá el siguiente número en la pantalla y el cursor se situará en la posición correcta. Para terminar el modo automático se pulsa ESC dos veces. La numeración automática puede empezar con cualquier número de línea (pruebe, por ejemplo, con AUTO 100); también se puede especificar el incremento de una línea a otra (pruebe con AUTO 10,5). Esto es muy cómodo cuando se está copiando el listado de un programa y éste está numerado uniformemente.

La otra función es RENUM. Escriba RENUM y su programa actual será renumerado, partiendo de 10 y con incrementos de 10. Se puede elegir a partir de qué línea debe empezar la renumeración, qué número debe tener la primera línea renumerada, y qué incremento debe hacer entre líneas sucesivas. Por extraño que parezca, hay muchos ordenadores que no realizan estas funciones.

3

Variaciones sobre el mismo tema

Hasta ahora nuestro trabajo se ha limitado a escribir números y palabras en la pantalla. Ese es uno de nuestros principales objetivos, desde luego, pero también debemos dedicar cierta atención a algunos de los procesos que tienen lugar antes de que los textos puedan aparecer en la pantalla. Uno de ellos es la asignación o definición de variables. Consideremos el programa de la figura 3.1. Escríbalo, ejecútelo y compare lo que ha escrito en el programa con lo que aparece en la pantalla. La primera

```
10 CLS
20 X=23
30 PRINT "2 POR"X"ES";2*X
40 X=5
50 PRINT "X ES AHORA"X
60 PRINT "Y EL DOBLE DE"X"ES";2*X
```

Fig. 3.1 Definición de variables. La letra X representa un número.

línea que contiene una instrucción de escritura es la 30. Su efecto es que en la pantalla aparezca

2 POR 23 ES 46

y sin embargo en la línea 30 no hemos escrito los números 23 ni 46. Esto se debe a que la letra X es una especie de código que representa, en este caso, el número 23. Un código de este tipo es lo que denominamos *variable* o *nombre de variable*.

La línea 20 define la variable X y le asigna el valor 23. Esto significa que siempre que utilicemos X (sin encerrarla entre comillas) el ordenador operará con el número 23. Puesto que X es un solo carácter y 23 tiene dos dígitos, hemos conseguido un ahorro de memoria. El ahorro sería más notable si hubiéramos definido, por ejemplo, X = 2174.3256. La línea 30 demuestra que efectivamente para el ordenador la letra X es el número 23: cada vez que se encuentra la X fuera de comillas, escribe 23, y en lugar de la expresión 2 * X escribe 46. Pero la X no tiene que seguir teniendo el mismo valor para siempre. De hecho, en la línea 40 le asignamos un valor dis-

tinto, el 5; las líneas 50 y 60 demuestran que el ordenador ha entendido el cambio.

Ésta es la razón por la que la X se llama «variable»: podemos hacerla variar a voluntad. No obstante, mientras no la cambiemos conservará el valor original. Incluso después de ejecutar el programa anterior, y siempre que no haya añadido o suprimido líneas, puede escribir PRINT X (ENTER) y verá el valor de X en la pantalla. A continuación de una palabra o frase escrita entre comillas se puede poner el nombre de una variable, pero *no una expresión* (tal como 2*X o X − 7). Por ejemplo, si escribe «ES» 2*X en el programa, provocará un mensaje de error. Por esta razón muchos programadores ponen sistemáticamente el punto y coma después de cerrar las comillas cuando éstas van seguidas tanto de una variable como de una expresión.

Volviendo a las variables, en lugar de una sola letra se puede utilizar un verdadero nombre. La única condición es que debe empezar por una letra, mayúscula o minúscula; los restantes caracteres pueden ser otras letras, hasta formar si se desea una palabra, o dígitos, pero no espacios, ni signos aritméticos (+, −, *, /) ni signos de puntuación. Así pues, son válidos nombres tales como TOTAL, ELRESTO o R2D2. Tenga cuidado con el siguiente detalle: el CPC464, como otros ordenadores, *no distingue mayúsculas de minúsculas.* Si asigna a la variable pan el número 45 y más tarde a la variable PAN el número 88, tanto pan como PAN tendrán el mismo valor (el último asignado), 88 en este caso. Las palabras reservadas (PRINT, NEXT, RUN, LIST, etc.) no pueden ser nombres de variables (se cometería error de sintaxis). Algunos ordenadores no permiten ni siquiera que los nombres de las variables *contengan* palabras reservadas, de forma que, por ejemplo, ANALISTA no puede ser el nombre de una variable. El CPC464 es más tolerante; se limita a exigir que los nombres de las variables no sean idénticos a ninguna palabra reservada. Si le pide que escriba el valor de una variable que no haya sido definida (es decir, a la que no se le haya asignado ningún valor), escribirá el número 0, pero no emitirá mensaje de error, que es lo que hacen otras máquinas.

También se pueden utilizar nombres para representar palabras y frases (o sea, cadenas literales). La diferencia con respecto a las variables numéricas es que al final del nombre de una variable literal hay que poner el signo de dólar ($). Si N es el nombre de una variable numérica, N$ (que se lee «ene-dólar») es el de una variable literal. El ordenador las trata como perfectamente distintas e independientes que son. La forma de definirlas también es diferente. Cuando se asigna un valor a una variable numérica, el número no se escribe entre comillas. Cuando se define una variable literal, la cadena a la que se iguala sí se escribe entre comillas. Más adelante estudiaremos otros métodos de asignación. En el siguiene ejemplo vamos a utilizar nombres de variables largos, entendiendo por tales los que constan de más de dos letras. Los nombres cortos ahorran espacio en la memoria, mientras que los largos ayudan a recordar el significado de la variable. Cuando se dispone de tan abundante memoria como en el CPC464, se puede ser generoso en la elección de nombres para las variables.

CADENAS LITERALES

En el programa de la figura 3.2 se utilizan *variables literales*, es decir, variables cuyo

```
10 CLS:NOMBRES$="CPC464"
20 PRIMERO$="EL EXCELENTE":ULTIMO$="TRABAJANDO"
30 PRINT PRIMERO$;" ";NOMBRE$;" ";ULTIMO$
40 PRINT "AQUI ESTA EL "; NOMBRE$
50 PRINT PRIMERO$;" ";NOMBRE$;" EN ACCION"
```

Fig. 3.2 Utilización de variables literales. Sus nombres terminan siempre en el signo de dólar.

«valor» es una sucesión de caracteres. En las líneas 10 y 20 se definen las variables; en las líneas 30 a 50 se muestra cómo se utilizan los nombres. Observe que se pueden mezclar los nombres de las variables, sin comillas a sus lados, con texto ordinario, que sí tiene que estar entre comillas. Hay que tener cuidado con esto porque puede resultar confuso. Observe en las líneas 30, 40 y 50 cómo hemos dejado espacios entre palabras.

Los signos de punto y coma no son imprescindibles para mezclar texto entre comillas con variables literales, siempre que no intervengan expresiones numéricas. Si omite los puntos y comas el efecto es el mismo.

En la figura 3.3 se muestra otro ejemplo, esta vez asignado a las variables UNA$ y OTRA$ frases largas. Si solamente fuésemos a imprimir el texto una vez, no tendría mucho sentido definir estas variables. En cambio, si una frase aparece varias veces en un programa, asignarla a una variable ahorra tener que mecanografiarla repetidamente. Obsérvese que en la línea 40 las dos variables están separadas por un punto y coma. Esta separación no es necesaria para el programa, pero ayuda al usuario a la hora de leer el listado.

```
10 CLS:UNA$="EL NUEVO ORDENADOR"
20 OTRA$="QUE OFRECE INFORMATICA SERIA A
   PRECIO MODERADO"
30 PRINT "EL CPC464:"
40 PRINT UNA$;OTRA$
50 PRINT UNA$+OTRA$
```

Fig. 3.3 Asignación de frases de varias palabras a variables literales.

Cuando se usa el punto y coma como separador entre variables, o aunque se escriban las variables una detrás de otra sin separador, la escritura en la pantalla se realizará tratando de evitar que se partan palabras. Si la cadena OTRA$ se escribiera inmediatamente después de UNA$, la palabra «informática» quedaría partida, según muestra la línea 40. Por supuesto, si la cadena total es más larga que la anchura de línea, esta forma de unir las cadenas no evitará la partición de palabras. En la línea 50 se unen las dos cadenas mediante el signo +.

MÁS SOBRE CADENAS

Puesto que todo nombre de variable literal ha de terminar en $, una variable tal como A$ no puede ser confundida con la variable numérica A. De hecho, se pueden utilizar ambos nombres en un programa sin provocar confusión (al menos para el ordenador). En la figura 3.4 se ilustra la diferencia. En las líneas 10 y 20 se definen

```
10 A=2:B=3
20 A$="2":B$="3"
30 CLS
40 PRINT A,B
50 PRINT A$,B$
60 PRINT A"POR"B" ES";A*B
70 PRINT A$" POR "B$" ES IMPOSIBLE!"
```

Fig. 3.4 Las variables literales y numéricas se parecen en este listado, pero para el ordenador son diferentes.

las variables numéricas A y B y las variables literales A$ y B$. Cuando las escribimos mediante las líneas 40 y 50, no se aprecia diferencia entre la A y la A$, ni entre la B y la B$, salvo que el hecho de que las variables numéricas van precedidas de un espacio. La diferencia entre unas y otras se hace en cambio bien patente cuando el ordenador intenta hacer cálculos aritméticos. Puede multiplicar variables numéricas por la misma razón por la que puede multiplar números, pero no puede multiplicar variables numéricas, tanto si los caracteres que las componen son letras como si son números. Dicho de otra forma, se puede multiplicar el número 2 por el número 3, pero no se puede multiplicar la cadena literal «2» por la cadena literal «3». Si en la línea 70 intentásemos realizar con las variables literales alguna operación aritmética, el ordenador respondería con un mensaje de error y el programa se detendría. El mensaje sería «Type mismatch in 70» («incongruencia de tipos en 70»); su significado es que se ha tratado de hacer con cadenas literales una operación que sólo se puede realizar en números. Más adelante veremos que hay ciertas operaciones que son específicas de las cadenas y que, por lo tanto, no pueden ser realizadas con números; si se intenta tal cosa, el mensaje de error que se obtiene es el mismo. La razón de esta especificidad de las operaciones radica en la distinta forma en que el ordenador almacena los dos tipos de variables, que está diseñada para facilitar a la máquina las operaciones que deba realizar con unas u otras.

Hay una operación que utiliza un signo aritmético y que sin embargo sólo se puede realizar con cadenas literales. El signo es el +, pero no entendido en su sentido aritmético. El programa de la figura 3.5 ilustra la acción de esta operación, denominada unión de cadenas o *concatenación*. Como se puede ver en las líneas 40 y 60, no se parece en nada a la suma aritmética. En la línea 50 se asignan a las variables

```
10 A$="MATESANZ"
20 B$="VILLANUEVA"
30 CLS
40 PRINT "LLAMEME SENCILLAMENTE "+A$+"-"+B$+" ,
   DIJO"
50 PRINT:A$="123":B$="456"
60 PRINT"LA CADENA GLOBAL ES ";A$+B$
70 PRINT"LA SUMA SERIA"; 579
```

Fig. 3.5 Concatenación o unión de cadenas, operación distinta de la suma aritmética.

cadenas compuestas por dígitos. En la línea 70 se muestra **cuál** sería el resultado si las variables fuesen numéricas. La concatenación proporciona un medio para generar cadenas compuestas ahorrando trabajo de mecanografía. Consideremos el programa de la figura 3.6. En la línea 10 se definen las variables A$ y B$ como una especie de «adornos» para utilizar en un título. El título en sí se define en la línea 20. La línea 40 escribe la cadena final.

```
10 A$="***":B$="###"
20 S$="EL NUEVO CPC464"
30 CLS
40 PRINT TAB(8)A$+B$+S$+B$+A$
```

Fig. 3.6 Utilización de la concatenación para construir un título con orla.

Relacionada con la concatenación, hay una instrucción que sirve para formar una cadena consistente en un solo carácter repetido cuantas veces se desee. Se trata de una instrucción STRING$, que debe ir seguida de dos parámetros entre paréntesis. El segundo parámetro es el carácter que se debe repetir; el primero es el número de repeticiones deseadas. Por ejemplo, si se define G$ = STRING$(20,«$»), la cadena G$ será una sucesión de 20 signos de dólar. En la figura 3.7 se da un ejemplo de programación de un título con orla.

```
10 CLS
20 A$=STRING$ (10,"*")
30 B$=STRING$ (5,"#")
40 PRINT:PRINT A$+B$+"TITULO"+B$+A$
```

Fig. 3.7 Ejemplo de utilización de STRING$ para formar una cadena de caracteres idénticos repetidos. Véase en la figura 8.10 otra aplicación de la instrucción STRING$.

ENTRADAS

Hasta ahora, cada vez que hemos querido visualizar texto en la pantalla hemos tenido que incluirlo previamente en el programa. Pero no tenemos que resignarnos a esta limitación, ya que el ordenador nos permite introducir datos, tanto numéricos como literales, en el programa durante la ejecución de éste. Una operación de este tipo es lo que se denomina «entrada». La instrucción de BASIC que realiza esta función es INPUT.

```
10 CLS
20 PRINT "ESCRIBA SU NOMBRE"
30 INPUT NOMBRE$
40 CLS:PRINT:PRINT
50 PRINT "BUENOS DIAS ";NOMBRE$
```

Fig. 3.8 Utilización de la instrucción INPUT. La instrucción 50 escribe en la pantalla el nombre tecleado.

En la figura 3.8 se da un ejemplo de utilización de esta instrucción. Como el programador no conoce de antemano el nombre del usuario, no podía incluirlo explícitamente en el programa. Al ejecutar este programa, en la pantalla aparece lo siguiente:

Escriba su nombre

En la línea siguiente a este mensaje se puede ver un signo de interrogación. Esto indica que el ordenador está esparando que se escriba algo y a continuación se pulse ENTER. Mientras tanto, el programa estará detenido en la línea 40. Al teclear el nombre no es necesario poner comillas; escríbalo tal como quiera verlo luego escrito en la pantalla. Al pulsar ENTER, el nombre que ha tecleado es asignado a la varible NOMBRE$. Este método de asignación a una variable literal no requiere el uso de comillas. Ahora se reanuda la ejecución del programa; la línea 40 borra la pantalla y avanza la posición de escritura a la tercera línea de la pantalla. La línea 50 escribe el texto programado en la línea 20 y su nombre a continuación. Por supuesto, en lugar de su nombre podría haber tecleado Pato Donald o Capitán Trueno o cualquier otra cosa. El ordenador no tiene forma de saber si se le está diciendo la verdad. Incluso aunque no teclee nada, y se limite a pulsar ENTER, el programa continuará, sin asignar ninguna cadena a la variable NOMBRE$.

La instrucción INPUT sirve también para la introducción de datos numéricos. En la figura 3.9 se da un ejemplo. Este programa es similar al anterior. Después de escribir el signo de interrogación, el ordenador espera que el usuario teclee un número y pulse ENTER. La acción de pulsar ENTER hace que el número tecleado sea asignado a la variable numérica N. La línea 40 prueba que el ordenador ha captado el número introducido. Cuando se utiliza INPUT seguida del nombre de una varia-

```
10 CLS:PRINT "ESCRIBA UN NUMERO, POR FAVOR"
20 INPUT N
30 CLS:PRINT
40 PRINT N"POR 2 ES";2*N
```

Fig. 3.9 INPUT aplicada a una variable numérica. El dato que se introduzca tiene que ser un número.

variable numérica, lo que se teclee antes de pulsar ENTER tiene necesariamente que ser un número. Si alguno de los caracteres tecleados no es un dígito, el ordenador se niega a aceptar el dato y emite un mensaje de error: «Redo from start» (vuelva a empezar). A diferencia de la mayor parte de los errores, éste no detiene el programa, sino que da al usuario otra oportunidad para introducir el dato correcto. En cambio, si después de INPUT se escribe el nombre de una variable literal, el ordenador aceptará *cualquier cosa* que se teclee; el error se producirá más tarde si se intenta realizar operaciones aritméticas con una cadena literal.

```
10 CLS
20 INPUT "POR FAVOR, ESCRIBA SU NOMBRE ";NOMBRE$
30 PRINT
40 PRINT "ENCANTADO DE CONOCERLE, ";NOMBRE$
```

Fig. 3.10 Utilización de INPUT para escribir una frase que solicita el dato.

La instrucción INPUT se puede utilizar de forma que parezca que el ordenador está atento a lo que el usuario teclea. En la figura 3.10 se ilustra otra forma de utilizar INPUT. La frase de solicitud del dato está entre comillas, seguida de un punto y coma y del nombre de la variable. La línea 20 de este programa equivale a las dos líneas siguientes:

```
15 PRINT "POR FAVOR, ESCRIBA SU NOMBRE",
20 INPUT NOMBRE$
```

En este caso el signo de interrogación y el cursor aparecen en la *misma línea* que la frase. La respuesta del usuario también estará en la misma línea, a menos que sea tan larga que no quepa en ella y tenga que continuar en la siguiente. En una instrucción como ésta se puede poner la coma en lugar del punto y coma. Pruébelo. La coma hace desaparecer el signo ?, pero no el cursor.

La instrucción INPUT puede ir seguida de varios nombres de variables, y se pueden mezclar variables literales con variables numéricas en una misma instrucción. En el programa de la figura 3.11 se captan dos variables en una misma instrucción INPUT. Al llegar a la línea 20, el ordenador escribe la frase y espera hasta que el usuario introduzca los dos datos: primero una cadena literal y luego un número.

```
10 CLS
20 INPUT "POR FAVOR, NOMBRE Y NUMERO: ";
   NOMBRE$,NUMERO
30 PRINT:PRINT
40 PRINT "EL NOMBRE ES ";NOMBRE$
50 PRINT "EL NUMERO ES";NUMERO
```

Fig. 3.11 Dos variables en una instrucción INPUT.

Sólo hay una forma correcta de hacer esto: escribir un nombre, después una coma, después un número y finalmente pulsar ENTER. Si pulsa ENTER después de teclear el nombre y antes de teclear el número, provocará el mensaje «Redo from start»; entonces tendrá que volver a teclear el nombre y luego el número (y pulsar ENTER). Las líneas 40 y 50 escribirán en la pantalla el nombre y el número introducidos. De lo expuesto se deduce que nunca se puede teclear nada que contenga una coma en respuesta a una instrucción INPUT. Por ejemplo, si la instrucción fuera INPUT NOMBRE$, no podría teclear ALVAREZ, JUAN. El ordenador trataría las dos palabras como entras distintas y sólo aceptaría ALVAREZ; además emitiría el mensaje «Redo from start». Los ordenadores son muy quisquillosos en estos detalles, y cada marca tiene su manía. No obstante, hay una instrucción que sí permite la introducción de comas: se trata de LINE INPUT. Esta instrucción se utiliza exactamente igual que INPUT.

```
10 CLS
20 INPUT "POR FAVOR, CUATRO NUMEROS:";A,B,C,D
30 PRINT
40 PRINT "SU SUMA ES";A+B+C+D
```

Fig. 3.12 Instrucción INPUT que capta cuatro números.

Veamos otro ejemplo. El programa de la figura 3.12 solicita cuatro números. También en este caso, los datos han de ir separados por comas; ENTER se pulsa sólo después de teclar los cuatro. Los números son asignados a las cuatro variables. La línea 40 escribe su suma.

LECTURA DE DATOS

Hay otra forma de introducir datos en un programa durante la ejecución. Consiste en leer los datos de una lista y utiliza dos instrucciones: READ (leer) y DATA (datos). La instrucción READ, cada vez que se ejecuta, hace que el programa lea un elemento de la lista. El programa identifica la lista porque ésta va precedida de

la palabra DATA. Los elementos de la lista tienen que estar separados por comas. Cada vez que se lee un elemento, el «puntero» pasa a señalar el elemento siguiente, el cual queda dispuesto para ser leído por la siguiente instrucción READ.

En el capítulo 5 volveremos a ocuparnos de estas dos instrucciones. Ahora vamos a ver un ejemplo sencillo, el de la figura 3.13. La línea 20 lee un dato, el primero de la lista (línea 70). La línea 30 lo escribe; el punto y coma del final impide que el cursor salte a la línea siguiente. El punto y coma de la línea 40 tiene el mismo efecto. La línea 50 lee el nombre que figura como segundo elemento en la lista de la línea 70. Este nombre es asignado a la variable NOMBRE$; la línea 60 lo escribe. La línea 70 contiene la lista de datos. Obsérvese que la cadena literal también está escrita sin comillas; si hubiéramos puesto comillas, el ordenador las ignoraría y no emitirá mensaje de error.

```
10 CLS
20 READ J
30 PRINT "EL PERIFERICO";J;
40 PRINT "ES UN ";
50 READ NOMBRE$
60 PRINT NOMBRE$
70 DATA 5,LECTOR DE DISCO
```

Fig. 3.13 Utilización de READ y DATA para suministrar datos a un programa.

Siempre hay que ser extremadamente cuidadoso con la congruencia entre los datos de las listas de DATA y las variables de READ. Si una instrucción READ requiere un dato numérico (por ejemplo, READ A), el elemento de la lista que va a ser leído tiene que ser un número. De lo contrario, el ordenador emitirá un mensaje de error y el programa se detendrá. El mensaje indicará que el defecto se encuentra en la línea de DATA y dará una oportunidad de corregirlo. (Quizá sea éste un buen momento para leer el apéndice sobre edición.) Si la variable de la instrucción READ es literal (por ejemplo, READ A$), la naturaleza del elemento leído es indiferente (es decir, no importa que sea literal o numérico). Recuerde, no obstante, que si un número ha sido leído como cadena literal, no será posible realizar con él ninguna operación aritmética. Observe que hay que dejar un espacio después de la palabra DATA y que los elementos de la lista tienen que estar separados por comas.

Las instrucciones READ ... DATA son útiles cuando se tiene una larga lista de datos que deben ser leídos repitiendo una instrucción READ. Los datos que se deben introducir en el programa de esta forma son los que el programa necesita siempre que es ejecutado. Los datos verdaderamente variables, que cambian con cada ejecución del programa, deben ser captados mediante instrucciones INPUT.

NÚMEROS

El trabajo que hemos desarrollado hasta ahora habrá convencido al lector de que los ordenaores hacen bastante más que cálculos numéricos. Pero, por supuesto, hay aplicaciones en las que la posibilidad de manipular números es fundamental. Si el lector piensa utilizar su ordenador en aplicaciones científicas o de ingeniería, la capacidad de la máquina para realizar calculos será mucho más importante que si pensara dedicarlo a proceso de textos, o incluso a contabilidad. Es tiempo, pues, de dar un repaso a las posibilidades de cálculo del CPC464. Será un repaso somero, porque no tenemos espacio para analizar detalladamente todas las funciones matemáticas de que esta máquina está dotada. En general, si el lector sabe en qué consisten funciones tales como «sen» o «exp», no tendrá ninguna dificultad a la hora de incluir estas funciones y operaciones en sus programas. En caso contrario, puede perfectamente omitir la lectura de los apartados que traten estas cuestiones.

```
10 CLS
20 X=5:PRINT"X ES"X
30 PRINT
40 X=X+1
50 PRINT "HA CAMBIADO!"
60 PRINT "X ES AHORA"X
```

Fig. 3.14 Operación de incrementar, utilizando el signo = con el significado de «se convierte en».

La operación más sencilla y elemental es la de contar. Nociones relacionadas son las de *incrementar* (contar hacia arriba) y *decrementar* (contar hacia abajo). Incrementar un número es sumarle 1. Decrementar un número es restarle 1. En BASIC estas operaciones se programan de una forma que puede parecer confusa, como muestra el programa de la figura 3.14. La línea 20 asigna a X el valor 5 y escribe ese valor de X. La línea 40 incrementa X; la extraña instrucción X = X + 1 hace que el nuevo valor de la variable sea igual al que tenía antes más 1. El resto del programa demuestra que la operación de incrementar X ha sido correctamente realizada.

```
10 CLS
20 X=5:PRINT"X ES"X
30 PRINT
40 X=2*X+4
50 PRINT "HA CAMBIADO!"
60 PRINT "X ES AHORA"X
```

Fig. 3.15 Modificación de una variable mediante una expresión algebraica.

El uso del signo = con el sentido de «se convierte en» es algo a lo que el lector tendrá que ir acostumbrándose. También podríamos escribir la instrucción

$$X = X - 1$$

cuyo efecto sería decrementar X, es decir, hacer X igual a su antiguo valor menos 1. También podríamos utilizar X = X * 2 para duplicar X, o X = X/3 para dar a X un valor igual a la tercera parte del que tenía antes. En la figura 3.15 se modifica X por multiplicación y suma.

```
10 CLS:X=2.5
20 PRINT "X ELEVADO AL CUADRADO ES";X↑2
30 PRINT
40 PRINT "SU RAIZ CUADRADA ES";SQR(X)
50 PRINT
60 PRINT "SU LOGARITMO NATURAL ES";LOG(X)
70 PRINT "Y SU LOGARITMO DECIMAL ES";LOG10(X)
```

Fig. 3.16 Algunas funciones matemáticas.

En la figura 3.16 se ilustra la utilización de varias funciones matemáticas. Una función es una instrucción que actúa sobre un número para producir otro número. La línea 10 asigna el valor 2.5 a X. La línea 20 escribe el cuadrado de X, esto es, X multiplicado por X. Esto se programa escribiendo X↑2; el carácter que el CPC464 utiliza para esta función está en la tecla del signo de libra esterlina. Para obtener la raíz cuadrada de un número utilizamos la instrucción SQR(). Otra forma posible es X↑.5, pero SQR(X) es más fácil de escribir (y de recordar, para los usuarios de habla inglesa). Para otras raíces se utilizan potencias fraccionarias; por ejemplo, la raíz cúbica es X↑(1/3). LOG(X) da el logaritmo natural o neperiano. El logaritmo decimal se obtiene con LOG10(X).

En la figura 3.17 se da una relación de las funciones matemáticas del CPC464, junto con una breve descripción de cada una de ellas. Algunas de estas funciones sólo tendrán utilidad en aplicaciones científicas, técnicas o estadísticas. Sin embargo, otras serán necesarias en las situaciones más inesperadas, como por ejemplo en la programación de gráficos. En la figura 3.17 se indica también el orden de prioridad de las operaciones; esto sirve para saber qué se obtiene cuando se escribe, por ejemplo, 3 + 4 * 5; en este caso, 23, ya que la multiplización se realiza siempre antes que la suma.

PRECISIÓN EN LOS CÁLCULOS

Uno de los problemas de los ordenadores pequeños es la precisión con la que realizan los cálculos. Como el lector sabe, la fracción 1/3 no se puede expresar con

---

ABS(X) Convierte los números negativos en positivos.

ATN(X) Da el ángulo (en radianes) cuya tangente es X.

BIN$(X,Y) Convierte el número decimal X en binario, y lo completa con ceros por la izquierda hasta hacerlo de longitud Y.

CINT(X) Convierte X en entero, redondeándolo si es necesario.

COS(X) Da el coseno del ángulo X (en radianes).

CREAL(X) Convierte X en «real».

DEG Interpreta los ángulos en grados.

EXP(X) Da el valor del número *e* elevado a X.

FIX(X) Toma la parte entera de X, ignorando la decimal.

FRE(X) Da el espacio de memoria libre.

HEX$(X) Convierte el número decimal X en hexadecimal.

INT(X) Da la parte entera de X, redondeando si X no es entero.

LOG(X) Da el logaritmo natural de X.

LOG10(X) Da el logaritmo decimal de X.

MAX(X,Y,Z,...) Da el valor máximo de los incluidos en la lista X,Y,Z,...

MIN(X,Y,Z,...) Da el valor mínimo de los incluidos en la lista X,Y,Z,...

PI Da el número π, relación de la longitud al diámetro de la circunferencia.

RAD Interpreta los ángulos en radianes.

RAMDOMIZE(X) Inicializa una nueva sucesión de números aleatorios.

RND(X) Da un número aleatorio comprendido entre 0 y 1.

ROUND(X,Y) Redondea el número X con el número de dígitos especificados por Y.

SGN(X) Da +1 si X es positivo; −1 si X es negativo; 0 si X es 0.

SIN(X) Da el seno del ángulo X (en radianes).

SQR(X) Da la raíz cuadrada de X.

TAN(X) Da la tangente del ángulo X (en radianes).

UNT(X) Traslada el número X, que debe estar comprendido entre 0 y 65536, al intervalo −32768, +32767.

Orden de prioridad

Para las operaciones aritméticas, el orden es MDSR, es decir, multiplicación y división, seguidas de suma y resta. El orden completo es

1. Potenciación ↑.
2. Multiplicación y división.
3. Suma y resta.
4. Comparación, utilizando = ⟨⟩.
5. AND lógico.
6. OR lógico.
7. NOT lógico.

Fig. 3.17 Funciones numéricas en el CPC464. (No se preocupe si no sabe para qué sirve alguna de ellas; si es así, seguramente no las necesita.)

exactitud mediante ningún número decimal. La aproximación a su valor verdadero será tanto mejor cuanto mayor sea el número de dígitos con que lo escribamos. Así, 0.33 es más preciso que 0.3, y 0.333 es aún más preciso. El ordenador convierte casi

```
10 CLS
20 PRINT 1/3,2/3
30 PRINT 1/11,10/11
40 PRINT 1/3+2/3,1/11+10/11
```

Fig. 3.18 Ilustración de cómo maneja el ordenador los números «difíciles». Los números muy grandes o muy pequeños son convertidos a *notación científica.*

todos los números con los que trabaja en una fracción y un multiplicador. La fracción no es decimal, sino de una forma especial denominada *fracción binaria.* Esta conversión raramente es exacta, y es particularmente torpe para números tales como 1, 10, 100, ... y 0.1, 0.01, 0.001, ..., es decir, para las potencias de 10. Para no tener que escribir cosas como 3 − 2 = 0.9999999, el ordenador redondea los números de este tipo, hacia arriba o hacia abajo, antes de escribirlos. No todos los ordenadores son tan hábiles en este aspecto como el CPC464. La figura 3.18 muestra cómo maneja el CPC464 fracciones tales como 1/3, 2/3, 1/11 y 10/11. Los números que escribe en la pantalla han sido redondeados a 9 decimales, pero internamente tiene que almacenarlos con precisión mucho mayor. El redondeo se efectúa de forma que la suma de fracciones dé el resultado correcto.

Por la forma en la que el ordenador escribe la fracción 1/11, el lector puede observar que hay otra manera de escribir los números. Si ha trabajado en algún área en la que se utilicen números muy grandes o muy pequeños (física, química, ingeniería), esto le será familiar. Por si no lo ha visto antes, esta forma de escribir los números se llama forma estándar o *notación científica.* Un número en notación científica consiste en un número comprendido entre 0 y 10 multiplicado por una potencia entera de 10. Consideremos, por ejemplo, el número 132000. Si desplazamos el punto decimal cinco lugares hacia la *izquierda* (lo que equivale a *dividir* por 10 elevado a 5), el número resultante es 1.32. Para volver al valor original, tendremos que multiplicar este número por 10 elevado a 5, o sea, por E5, pues ésta es la forma en que los ordenadores escriben las potencias de 10. En resumen, el número 132000 se puede escribir en la forma 1.32E5.

Probemos ahora con un número pequeño: 0.00036. Este número es igual a 3.6 por 10 elevado a menos 4, o sea, 3.6E-4; el signo «menos» aparece porque hemos tenido que desplazar el punto decimal cuatro lugares hacia *la derecha.* El CPC464, como casi todos los ordenadores pequeños, acepta y escribe números en esta notación. La conversión es automática. El CPC464 escribe los números en notación científica solamente cuando le es necesario, es decir, cuando en notación ordinaria requerirían más de nueve dígitos.

La mayor parte de los ordenadores permite el manejo, mucho más preciso y económico en lo que a memoria se refiere, de una clase especial de números: son las llamadas *variables enteras.* Para el CPC464, un entero es un número entero comprendido entre −32768 y +32767. El nombre de una variable entera ha de terminar con el signo %. Al asignar valores a varibles enteras, es necesario que el valor cumpla las limitaciones mencionadas. Por ejemplo, si intentamos hacer algo como

A% = 32800

provocaremos el mensaje de error «Overflow» (rebosamiento o sobrepasamiento). El programa de la figura 3.19 muestra cómo se deshace el ordenador de la parte decimal cuando el resultado de los cálculos con números enteros no es entero. En efecto, en la línea 40 se asigna a la variable entera X% el valor de la variable X, que es 3.7. Sin embargo, la línea 40 imprime 4. La parte decimal 0.7 ha sido redondeada hacia arriba, a 1. Esto es infrecuente, y afortunado, porque muchos ordenadores, en lugar de redondear, se limitan a tomar la parte entera; es decir, omiten 0.7 para dar 3.

```
10 CLS:X=3.7
20 PRINT "X ES UN NUMERO REAL IGUAL A"X
30 PRINT "X% ES UN ENTERO"
40 X%=X
50 PRINT "EL VALOR DE X% ES"X%
60 Y%=7/5
70 PRINT "7/5, CONSIDERADO COMO ENTERO, ES"Y%
```

Fig. 3.19 Las variables enteras ocupan menor espacio de memoria y pueden ser más precisas, pero no en la división.

La ventaja de utilizar variables enteras es doble. La primera es que los cálculos aritméticos (salvo la división) dan resultados exactos, sin necesidad de redondeo. La división es una excepción, como demuestra la línea 70 del programa de la figura 3.19. La otra ventaja de los enteros es que ocupan menor espacio en la memoria. Por otra parte, la velocidad de cálculo es mucho mayor con variables enteras que con cualesquiera otras. El CPC464 permite realizar la división entera utilizando las funciones y MOD (la tecla \ está junto a la /). La operación da la parte entera del cociente. Por ejemplo, si tecleamos

PRINT 7\3

el resultado que aparece en la pantalla es 2, porque, efectivamente, 7/3 es 2 más una parte decimal. La operación MOD da el resto de la división entera. Si tecleamos

PRINT 7 MOD 3

el resultado es 1, ya que 1 es el *resto* de dividir 7 por 3. Estas dos operaciones tienen su utilidad en los métodos de programación más avanzados, para los que no tenemos espacio en este libro.

Otra posibilidad especialmente útil en las aplicaciones matemáticas es la *definición de funciones.* El interés de la definición de funciones reside en que, con sólo teclearlas una vez, se las puede utilizar en el programa tantas veces como se desee. La figura 3.20 ilustra cómo se utiliza la instrucción DEF. El ejemplo es muy sencillo, y en práctica nunca se definiría una función solamente para esto. Las líneas 20 y 30 captan tres números. En la línea 40 se escribe el valor de un ente llamado FNsum

```
5 DEF FNSUM(A,B,C)=A+B+C
10 CLS
20 PRINT "POR FAVOR, INTRODUZCA TRES NUMEROS"
30 INPUT A,B,C
40 PRINT "SU SUMA ES";FNSUM(A,B,C)
```

Fig. 3.20 Definición de funciones.

(A,B,C); para que el ordenador conozca esta función, tiene que haber sido definida previamente en el programa. La instrucción de definición debe empezar con DEF FN; a continuación, el nombre elegido para la función; finalmente, entre paréntesis, las variables de la función (los parámetros). Todo esto tiene que estar en una sola línea de programa. Se puede considerar como si DEF FN proporcionara una fórmula que la máquina tiene que consultar cada vez que encuentre un FN en el programa. En este esjemplo hemos puesto la definición en la línea 5 para hacer que la función sea conocida de la máquina cuando la necesite. Si la hubiéramos definido, por ejemplo, en la línea 100, al ejecutar la línea 40 el ordenador habría emitido un mensaje de error de sintaxis.

```
10 DEF FNHIPOT(A,B)=SQR(A↑2+B.↑2)
20 CLS
30 INPUT "POR FAVOR, INTRODUZCA LOS CATETOS DE UN
   TRIANGULO RECTANGULO ";X,Y
40 PRINT "LA HIPOTENUSA ES";FNHIPOT(X,Y)
```

Fig. 3.21 Otra función definida, esta vez para calcular la hipotenusa de un triángulo rectángulo dados los valores de los catetos.

La figura 3.21 es otro ejemplo de definición de funciones. La definición se realiza al principio del programa. FNhipot calcula la raíz cuadrada de a↑2 + b↑2, pero en las líneas 30 y 40 las variables que representan los catetos son *x* e *y*. En realidad, DEF FN indica al ordenador qué debe hacer con un par de números; el nombre que se dé a los números carece de importancia. Los recursos de este tipo facilitan la labor de programación; a medida que vaya dquiriendo soltura aprenderá a utilizarlos con fluidez.

DEFINICIÓN DE TIPOS DE VARIABLES

En algunos programas ocurre que todas las variables utilizadas son de un mismo tipo. Se puede ahorrar tiempo de mecanografía recurriendo a otra versión de la instrucción DEF. Por ejemplo, DEFINT A-Z hace que todas las variables que em-

piecen por una letra de la A a la Z sean consideradas enteras, de modo que al teclear el programa no es necesario escribir A%, B%, etc., sino sólo A, B, etc. DEFSTR sirve para hacer literales todas las variables. DEFREAL para hacerlas reales. También se pueden mezclar los tipos; por ejemplo, DEFSTR A-I,O-Z:DEFINT J-N define como enteras todas las variables cuyo nombre empiece por J a N, y literales todas las demás.

4

Repetición de tareas

Una de las actividades para las que el ordenador está especialmente bien dotado es la repetición de un conjunto de instrucciones. Todos los ordenadores disponen de medios para llevar a cabo tales repeticiones. El CPC464 no es excepción; de hecho, maneja más instrucciones de repetición que los demás ordenadores de su nivel de precio, e incluso que otros mucho más caros. Describiremos en primer lugar la más sencilla de estas instrucciones: GOTO (ir a).

La acción de esta instrucción es precisamente lo que su nombre sugiere: ir a otro número de línea. Normalmente, los programas ejecutan las instrucciones en orden ascendente de los números de línea; es decir, empiezan por la línea de número más bajo y terminan en la línea de número más alto. La instrucción GOTO puede alterar este orden, pues permite saltarse algunas instrucciones o repetir la ejecución de otras. La palabra reservada GOTO debe ir seguida de un espacio y, a continuación, de un número de línea. La omisión del espacio provoca un mensaje de error.

```
10 PRINT "AMSTRAD LLENA LA PANTALLA"
20 GOTO 10
```

Fig. 4.1 Un bucle infinito muy sencillo. Para detenerlo, pulse ESC dos veces.

En la figura 4.1 se da un ejemplo de un sencillo mecanismo de repetición o *bucle*. La línea 10 es sencillamente una instrucción PRINT. Cuando ésta ha sido ejecutada, el programa pasa a la línea 20, que le ordena volver a la línea 10. Este bucle no termina nunca; es lo que denominamos un *bucle infinito*. Su efecto es escribir en la pantalla

AMSTRAD LLENA LA PANTALLA

una y otra vez hasta que se pulse la tecla ESC para interrumpir el bucle. Siempre que la máquina parezca haber entrado en un bucle infinito se la puede parar de esta forma; el programa se detiene, pero no completamente. Si a continuación se pulsa cualquier otra tecla, el programa se reanuda a partir de la situación en la que fue momentáneamente interrumpido. Más adelante veremos que ésta es una forma muy

útil de detectar errores en los programas. Si lo que se desea es detener definitivamente el programa, se debe pulsar la tecla ESC dos veces.

```
10 CLS:N=10
20 PRINT N
30 N=N+1
40 GOTO 20
50 REM - PULSE ESC ESC PARA TERMINAR
```

Fig. 4.2 Buble que utiliza un contador muy rápido. Para detenerlo, puse ESC dos veces.

Vamos a probar otro bucle en el que se realiza alguna actividad adicional. El programa de la figura 4.2 escribe en la pantalla un número cada vez que recorre el bucle, o sea, con cada *pasada por el bucle*. La línea 10 asigna el valor inicial 10 a la variable N. La línea 20 escribe este valor y la línea 30 lo incrementa. La línea 40 es la instrucción de retorno del bucle. El efecto global es visualizar en la pantalla un contador muy rápido. También en este caso, para interrumpir el bucle se pulsa ESC, lo que da una oportunidad para comprobar cómo está funcionando el programa. Si todo va bien, se pulsa culquier otra tecla; si se desea detener definitivamente el programa, se pulsa ESC por segunda vez.

Los bucles de este tipo son incontrolados, y por consiguiente nada útiles. No todos los ordenados ofrecen una alternativa; el CPC464 ofrece dos. Una de ellas es el bucle tipo FOR ... NEXT (para ... siguiente). Como su nombre sugiere, en estos bucles se utilizan dos palabras reservadas, FOR y NEXT. Las instrucciones que el bucle repite son las que incluyamos entre FOR y NEXT. La figura 4.3 da un ejemplo muy sencillo. La línea que contenga FOR tiene que incluir también la variable numérica que va a servir como contador, así como sus valores inicial y final. En este ejemplo, la variable numérica elegida es N, y sus valores extremos son 1 y 10. La instrucción NEXT está en la línea 40, de modo que lo que se repite con cada pasada del bucle es todo lo que haya entre las líneas 20 y 40.

```
10 CLS
20 FOR N=1 TO 10
30 PRINT "AMSTRAD, EL MAS PODEROSO!"
40 NEXT
```

Fig. 4.3 Bucle FOR ... NEXT para repetir un número determinado de veces.

En este ejemplo no hemos sido nada ambiciosos. Lo único que hemos incluido entre esas dos líneas es una instrucción PRINT; el programa escribe «AMSTRAD, EL MÁS PODEROSO» diez veces. En la primera pasada del bucle el valor de N es 1 y se

escribe la frase. Al llegar a la instrucción NEXT, el ordenador incrementa el valor de N (en este caso, de 1 a 2). A continuación comprueba si el nuevo valor de N excede del límite preestablecido, 10. Si no es así, se repite la línea 30. El proceso se repite hasta que el valor de N exceda de 10; más adelante volveremos sobre esta cuestión. El efecto de este programa es realizar la acción 10 veces.

```
10 CLS
20 FOR N=1 TO 10
30 PRINT "EL CONTADOR MARCA";N
40 FOR J=1 TO 1000:NEXT
50 CLS:NEXT
```

Fig. 4.4 Programa con bucles anidados, uno dentro del otro.

Los bucles no tienen por qué ser únicos. En la figura 4.4 se da un ejemplo de *bucles anidados*, es decir, de un bucle dentro de otro. En casos como éste a los bucles les llamamos *interno* y *externo*. El bucle externo empieza en la línea 20 y utiliza la variable N, con valores extremos 1 y 10. La línea 30 forma parte de este bucle externo; escribe el valor del contador N. La línea 40 es un bucle distinto e independiente. En él hay que utilizar como contador otra variable, y tiene que empezar y terminar antes del final del bucle externo. Para el contador de este bucle interno hemos elegido la variable J. Entre FOR y NEXT no hemos intercalado instrucciones; lo único que hace el bucle interno es, pues, perder tiempo, para permitir observar las acciones del bucle externo. La última instrucción del bucle principal (externo) es borrar la pantalla (línea 50). El efecto global del programa es visualizar en la pantalla un contador, suficientemente lento como para ser observable. En este ejemplo hemos indicado el final de cada bucle con NEXT. También podríamos haber puesto NEXT N en la línea 40 y NEXT J en la línea 50, pero no era necesario. De haberlo hecho así, el programa habría resultado algo más lento, lo que en este caso no tendría importancia. Si a continuación de NEXT se pone el nombre de la varible, hay que poner el correcto. Si no, el programa se detendrá después de emitir el mensaje de error «NEXT missing» (falta NEXT), lo que indicaría, en este ejemplo, que se han intercambiado las variables. El mismo mensaje se provocaría si se omitiera algún NEXT.

Ya en este momento se puede apreciar lo útil que puede ser un bucle FOR ... NEXT, pero aún tenemos que sacarle más partido. Para empezar, los bucles que hemos considerado hasta ahora contaban todos hacia arriba, incrementando la variable numérica. No siempre es esto lo que necesitamos. Para modificar la forma en que se actualiza el contador podemos añadir la palabra STEP (paso al final de la instrucción FOR. Una instrucción completa podría ser, por ejemplo,

FOR N = 1 TO 9 STEP 2

La sucesión de valores de N sería 1, 3, 5, 7, 9. Si omitimos STEP, el contador se incrementa de 1 en 1.

```
10 CLS
20 FOR N=10 TO 1 STEP -1
30 PRINT "QUEDAN";N;"SEGUNDOS"
40 FOR J=1 TO 800:NEXT
50 CLS:NEXT
60 PRINT"DESPEGUE!!"
```

Fig. 4.5 Programa que cuenta hacia atrás.

La figura 4.5 muestra un programa en el que el bucle principal tiene paso −1 y que, por consiguiente, cuenta hacia atrás. N tiene valor inicial 10 y el programa lo decrementa con cada pasada del bucle. La línea 40 es un retardo que hace que la cuenta atrás tenga lugar a velocidad más «humana». Ésta es una forma de retardar el contador. Si lo que queremos es acelerarlo, podemos utilizar una variable entera; por ejemplo, J% en lugar de J. Pero entonces el valor del paso no puede contener parte decimal (por ejemplo, STEP 0.1).

```
10 CLS
20 FOR N=1 TO 5
30 PRINT N
40 NEXT
50 PRINT"N ES AHORA";N
60 FOR N=5 TO 1 STEP -1
70 PRINT N
80 NEXT
90 PRINT"N ES AHORA";N
```

Fig. 4.6 Determinación del valor del contador cuando ha terminado el bucle.

En ocasiones, cuando manejamos bucles, tenemos que utilizar el valor del contador una vez terminado el bucle. Es necesario saber cuál será ese valor; la figura 4.6 nos da la solución. Este programa contiene dos bucles, uno que cuenta hacia arriba y el otro hacia abajo. Al final de cada bucle se escribe el valor de la variable utilizada como contador. Así vemos que N es 6 en la línea 50, después de completar el bucle FOR N = 1 TO 5, y que es 0 en la línea 90, después de completar el bucle FOR N = 5 TO 1 STEP −1. Cuando se va a aprovechar la variable del contador, se debe tener en cuenta que habrá sido modificada una vez más que las especificadas por la instrucción FOR. Si por razones de velocidad decide utilizar variables enteras, recuerde que los valores de éstas tienen sus limitaciones. No se puede permitir que una variable entera sea mayor que 32767 ni menor que −32768. Si el valor sobrepasa estos límites, se provoca el mensaje de error «Overflow».

Una de las características más interesantes de los bucles FOR ... NEXT consiste en que en su definición se pueden utilizar variables numéricas en lugar de números. En el ejemplo de la figura 4.7 a las variables A, B y C se les agignan valores en la forma habitual (línea 20), y luego se las incluye en las instrucción FOR (línea 30). Los límites son A y B; el paso es la expresión B/C. La regla es que para la instrucción FOR es válido todo lo que sea o represente un número, en sí mismo o como resultado de un cálculo.

```
10 CLS
20 A=2:B=5:C=10
30 FOR N=A TO B STEP B/C
40 PRINT N
50 NEXT
```

Fig. 4.7 Bucle en el que los valores límite y el paso son variables numéricas.

BUCLES Y DECISIONES

Ha llegado el momento de utilizar los bucles, y no simplemente demostrar cómo funcionan. Una aplicación sencilla consiste en totalizar números. Queremos escribir un programa que capte datos por el teclado y haga que el ordenador mantenga y visualice su suma total. Por lo que hemos aprendido hasta ahora, la solución es inmediata si sabemos de antemano cuántos números vamos a sumar. El programa de la figura 4.8 suma diez números.

```
10 TOTAL=0:CLS
20 PRINT TAB(8)"PROGRAMA TOTALIZADOR"
30 PRINT:PRINT"INTRODUZCA LOS NUMEROS UNO A UNO,
   SEGUN SE LOS VAYA PIDIENDO."
40 PRINT "EL PROGRAMA CALCULA LA SUMA TOTAL."
50 FOR N=1 TO 10
60 PRINT "POR FAVOR, EL NUMERO";N;
70 INPUT J:TOTAL=TOTAL+J
80 NEXT
90 PRINT:PRINT"LA SUMA TOTAL ES";TOTAL
```

Fig. 4.8 Programa que capta diez números y calcula su suma.

El programa empieza asignando el valor 0 a la variable «TOTAL». Esta es la variable que utilizaremos para guardar la suma de los números, luego ha de partir de

cero. El CPC464 pone a cero todas las variables cuando comienza la ejecución de un programa, pero es buena costumbre vigilar los valores iniciales de las variables cada vez que se las va a manejar. Dicho sea de paso, a esta variable no le podríamos haber llamado TO (lo que sería una abreviatura obvia) porque TO es una palabra reservada de BASIC. El ordenador emite el mensaje «Syntax error» si se utiliza una palabra reservada como nombre de una variable.

Las líneas 20 a 40 escriben en la pantalla instrucciones para el usuario. En la línea 50 se establece el bucle FOR ... NEXT, que hará que se repitan diez veces las acciones de las líneas 60 y 70. La línea 60 recuerda al usuario cuántos números lleva introducidos escribiendo el valor de N en cada pasada del bucle. La línea 70 capta un número, que es asignado a la variable J. Este valor se suma al valor actual de la variable «TOTAL» (segunda instrucción de la línea 70). La línea 80 reinicia el bucle. Al final del programa, la variable «TOTAL» contiene la suma de todos los números introducidos.

Todo esto está muy bien, ¿pero cuántas veces vamos a tener que sumar precisamente diez números? Preferible sería disponer de un programa que se detuviera cuando le hiciéramos una señal convenida, por ejemplo, introducir un número tal como el 0 o el 999. Un número de este tipo se denomina *finalizador*; tiene que ser algo evidentemente distinto de los datos «normales». Para un programa que sume números, es adecuado utilizar como finalizador el 0, porque no importa que el programa lo sume. En cualquier caso, necesitamos una forma de detectar la introducción del finalizador. Para ello sirve la instrucción IF (si).

IF tiene que ir seguido de una condición. Por ejemplo, IF N = 20 o IF NM$ = «ULTIMO». Después de la condición se puede poner la palabra THEN (entonces), y ésta debe ir seguida de lo que se quiere que el ordenador haga, todo ello en una misma línea. Por ejemplo, si queremos que el programa termine cuando se cumpla la condición, después de THEN podemos poner un número de línea; si el número es el de la última línea del programa, éste se detiene en cuanto se cumpla la condición.

```
10 CLS:PRINT TAB(10)"OTRO TOTALIZADOR"
20 PRINT:PRINT"ESTE PROGRAMA CALCULA LA SUMA DE LOS"
30 PRINT"NUMEROS HASTA QUE INTRODUZCA UN CERO."
40 TOTAL=0
50 INPUT"POR FAVOR, EL NUMERO":N
60 TOTAL=TOTAL+N
70 PRINT"HASTA AHORA LA SUMA ES";TOTAL
80 IF N<>0 THEN 50
90 PRINT"FIN"
```

Fig. 4.9 Programa totalizador que no puede utilizar un bucle NEXT ... FOR. La línea 80 decide si el programa debe continuar o terminar.

Si todo esto le suena a complicado, considere el ejemplo de la figura 4.9. En este caso no podíamos utilizar un bucle FOR ... NEXT porque no sabemos de antemano cuántos números se van a introducir. Así pues, hemos recurrido a la instrucción IF ... THEN para controlar el bucle. Las primeras líneas escriben en la pantalla instrucciones para el usuario. En la línea 40 inicializamos la variable «TOTAL». Cada vez que se introduce un número en respuesta a la línea 50, éste es sumado a TOTAL en la línea 60. La línea 70 escribe el valor actual de la suma. La línea 80 contiene el control del bucle. La instrucción IF hace la comprobación; en este caso determina si N no es cero. Si no lo es, reenvía el programa a la línea 50. El signo ⟨⟩ está compuesto por los signos «menor que» y «mayor que» y significa «distinto de». Después de THEN y antes del número de línea se puede poner la palabra GOTO, pero no es necesario.

El funcionamiento del control del bucle es el siguiente. Si el número captado por pa línea 50 no es cero, la condición que sigue a IF en la línea 80 se cumple y el programa continúa en la línea 50. El bucle se repite hásta que se introduzca un cero. Entonces la condición de 80 no se cumple y el programa pasa a la línea 90. Ésta anuncia el fin del programa; como no hay más líneas, el programa concluye. Si se pulsa ENTER sin haber tecleado antes un número, el programa actúa como si se hubiera introducido un cero. No todos los ordenadores se comportan tan sensatamente como éste.

En este ejemplo sólo se realiza una comprobación, con su IF. Pero podríamos haber programado un control de bucle en el que se determinase si el número introducido es un 0 o 999. Para ello la línea IF tendría que ser

```
IF N⟨⟩0 AND N⟨⟩999 THEN 50
```

Para que el programa salte a 50 ahora N tiene que cumplir dos condiciones. También podrámos utilizar la palabra OR:

```
IF X = 0 OR X = 999 THEN ...
```

En un programa totalizador como el anterior hay que evitar que el número 999 sea sumado. El control de bucle tendría que estar en otro sitio.

Signo	Significado
=	Igualdad
⟩	El número que está a la izquierda es mayor que el de la derecha
⟨	El número que está a la izquierda es menor que el de la derecha

Estos signos se pueden combinar de las siguientes formas:

⟨⟩	Números distintos
⟩=	Número de la izquierda mayor o igual que el de la derecha
⟨=	Número de la izquierda menor o igual que el de la derecha

Nota: Cuando se combinan los signos ⟨ y ⟩ con =, éste se tiene que escribir en último lugar. Una combinación tal como =⟩ da lugar a un mensaje de error.

Fig. 4.10 Signos matemáticos que se utilizan para comparar números y variables numéricas.

En la figura 4.10 se da una lista de los tipos de comprobaciones que se pueden realizar con la instrucción IF. Por comodidad se utilizan signos matemáticos, pero también tienen sentido aplicados a cadenas literales. De lo que hemos visto hasta ahora se puede deducir qué significado pueden tener ⟨⟩ y =; más adelante veremos cómo se utilizan ⟨ y ⟩.

ELSE

La instrucción IF ... THEN es una fórmula de comprobación muy útil tal como la conocemos. Pero existe otra versión aún más potente. Puede ir seguida de ELSE (si no) para realizar una segunda comprobación y tomar la decisión pertienente. Aclaremos esto con el ejemplo de la figura 4.11.

```
10 PRINT TAB(14)"CARA O CRUZ"
20 PRINT"(ESCRIBA E PARA TERMINAR)"
30 N=1+INT(2*RND(2))
40 IF N=1 THEN PRINT"CARA" ELSE PRINT"CRUZ"
50 PRINT"ESCRIBA E PARA TERMINAR"
60 INPUT A$:IF A$="E" THEN END ELSE 30
```

Fig. 4.11 Programa que simula el lanzamiento de una moneda.

Es un sencillo programa que simula el lanzamiento de una moneda. Las líneas 10 y 20 realizan el trabajo habitual. El bucle empieza en la línea 30, que es la línea fundamental para la simulación. RND significa «elegir un número al azar»; esta palabra, seguida de un número positivo entre paréntesis, genera aleatoriamente un número comprendido entre 0 y 1 (pero nunca exactamente igual ni a 0 ni a 1). Al multiplicar este número por 2 se obtiene un número comprendido entre 0 y 2. Tomando su parte entera con INT, se obtiene un número entero que puede ser 0 o 1. Si sumamos 1 a este número, el resultado puede ser 1 o 2, y esto es lo que necesitamos para decidir si el resultado del lanzamiento de la moneda ha sido cara o cruz. La comprobación se realiza en la línea 40: si N es 1, se escribe la palabra «CARA», y «CRUZ» en el otro caso. En este ejemplo hemos utilizando ELSE para elegir un segundo curso de acción. Esta misma palabra sirve en la línea 60 para decidir si se debe continuar o detener el programa. El lector puede complicar el programa haciendo que pida al usuario que adivine el próximo resultado y llevando la cuenta de aciertos y fallos.

La importancia de ELSE reside en que da una opción. Si se cumple la condición se realiza cierta acción (por ejemplo, escribir un mensaje); si no se cumple, se realiza una acción distinta. Más adelante veremos cómo programar comprobaciones más complejas. De momento vamos a estudiar otro tipo de bucles, más flexible en cuanto a la forma de terminarlo.

WHILE ... WEND

Muy pocos ordenadores domésticos ofrecen otras posibilidades de construir bucles que la simple FOR ... NEXT; algunos ni siquiera conocen la palabra ELSE. El CPC464, en cambio, está dotado de un tipo de bucle muy diferente: WHILE ... WEND (mientras ... fin de bucle). Esta instrucción facilita la programación de bucles, hasta el punto de que no es necesaria la instrucción GOTO, y ni siquiera IF ... THEN. La idea consiste en empezar el bucle con una condición, a continuación poner todas las líneas que se desee y terminar el bucle con WEND (palabra construida con la inicial de WHILE y la palabra END).

```
10 CLS:PRINT TAB(15)"OTRO TOTALIZADOR":TOTAL=0:N=1
20 PRINT:PRINT"INTRODUZCA LOS NUMEROS
   (0 PARA TERMINAR)"
30 WHILE N<>0
40 INPUT N
50 TOTAL=TOTAL+N
60 PRINT"LA SUMA ACTUAL ES";TOTAL
70 WEND
80 PRINT"FIN"
```

Fig. 4.12 Programa totalizador que utiliza un bucle del tipo WHILE ... WEND.

Un ejemplo aclarará estas ideas. El programa de la figura 4.12 es otra versión de un viejo conocido, el totalizador. Esta vez, al principio del programa no sólo hacemos TOTAL = 0, sino también N = 1. Esto es necesario debido a la forma de trabajar del bucle WHILE ... WEND. El principio del bucle es la línea 30, WHILE N⟨ ⟩0. Esto significa que el bucle será repetido mientras N sea distinto de cero. Sin embargo, en la primera pasada del bucle aún no se ha introducido ningún N; por ello hemos tenido que definir previamente la variable N con su valor cualquiera (distinto de cero, por supuesto). Si no hubiéramos hecho esta definición, el programa terminaría en cuanto llegase a la línea 30.

Las instrucciones internas del bucle ya nos son familiares. La instrucción nueva es la de la línea 70, WEND. Esta palabra señala el final del bucle, y transfiere el programa automáticamente a la línea 30. En ésta no son necesarias las comprobaciones del tipo IF ... THEN. Incluso se pueden anidar los bucles WHILE ... WEND, como muestra el ejemplo del manual, más complejo que éste. El único problema es que la comprobación se efectúa al principio del bucle; esto obliga a asignar con antelación un valor a toda variable que intervenga en la línea de WHILE.

Consideremos otro ejemplo, el de la figura 4.13. Esta vez utilizamos READ DATA dentro del bucle. El programa lee datos de una lista hasta que encuentra una X. En la línea 20 se asigna a A$ la cadena literal consistente en un espacio. El bucle empieza en la línea 30; la condición hace que el bucle se repita hasta que el valor

```
10 CLS
20 A$=" "
30 WHILE A$<>"X"
40 PRINT A$
50 READ A$
60 WEND
70 DATA ALEMANIA,FRANCIA,ITALIA,GRECIA,INGLATERRA
80 DATA DINAMARCA,SUECIA,X
```

Fig. 4.13 Otro ejemplo de bucle WHILE ... WEND, con READ en su interior.

de A$ sea la cadena «X». La acción de las instrucciones internas del bucle consiste en escribir A$ (un espacio en la primera pasada) y luego leer en la lista DATA otro valor de A$. La línea 60 es el punto de retorno del bucle, que envía el nuevo valor de A$ a la línea 30 para su comprobación. El efecto global del programa es escribir la lista de datos. ¡Muy ingenioso!

```
10 INPUT"TECLEE UN NUMERO DEL 1 AL 5";N
20 WHILE N<1 OR N>5
30 PRINT"INCORRECTO. SOLO DEL 1 AL 5, POR FAVOR."
40 INPUT N
50 WEND
60 PRINT"HA ELEGIDO EL"N
```

Fig. 4.14 Programación de una trampa para errores con un bucle WHILE ... WEND.

En la figura 4.14 se ilustra otra aplicación del bucle WHILE ... WEND: una trampa para errores. Una trampa para errores es la sección de un programa que se encarga de comprobar los datos tecleados (en lenguaje más informático, de validar los datos). Si el dato introducido es incorrecto, por ejemplo, por ser un número fuera del margen permitido, la trampa lo rechaza y vuelve a solicitar el dato correcto. Estos mecanismos son muy importantes cuando se quiere evitar que la introducción de un dato imprevisto detenga el programa con un mensaje de error. Para un experto (y el lector lo será cuando termine de leer este libro), tal detención no será problema, pues siempre podrá reanudar la ejecución con una orden GOTO. Para el inexperto, un mensaje de error puede ser la gota que colma el vaso.

En este ejemplo, al usuario se le pide que introduzca un número del 1 a 5. Si el número introducido es correcto, el programa termina con un mensaje de confirmación; si no lo es, el bucle entra en acción (haga la prueba). Esta acción consiste en escribir un mensaje de error controlado por el programa y en dar otra oportunidad para que el usuario introduzca un dato correcto. Éste es el objetivo de toda trampa

para errores; el bucle WHILE ... WEND es idóneo para esta tarea. Obsérvese que, a pesar de que en casi todos los ejemplos hemos manejado números, el bucle WHILE ... WEND funciona perfectamente con cadenas literales. Se pueden programar líneas tales como

WHILE NOMBRE$<>"X"

para controlar el fin de la captación de una serie de datos literales, o también

WHILE RE$<>"S" AND RE$<>"N"

para construir una trampa que sólo admita como respuesta «S» o «N».

RESPUESTA CON UNA SOLA TECLA

Hasta ahora nuestros programas han captado las respuestas del tipo S/N mediante instrucciones INPUT; esto requiere teclear la respuesta y luego pulsar ENTER. La ventaja de este método es que permite rectificar la respuesta, siempre que la rectificación se haga antes de pulsar ENTER. Para mayor agilidad, se puede utilizar la función INKEY$. Esta instrucción realiza una comprobación del teclado para determinar si se está pulsando una tecla. La comprobación es muy rápida, y la única forma de utilizarla es incluirla en un bucle que se repita hasta que efectivamente se pulse una tecla. En la figura 4.15 se ha programado un bucle para producir un efecto de «esperar hasta que esté preparado». El bucle WHILE ... WEND es muy sencillo: se repite mientras INKEY$ sea una *cadena vacía*, es decir, una cadena que no contenga ningún carácter. La cadena vacía se escribe en los programas escribiendo dos signos de comillas seguidos, sin nada en medio. Cada vez que se ejecuta INKEY$, el ordenador examina el teclado para averiguar si se está pulsando una tecla. Si no se está pulsando ninguna, INKEY$ es una cadena vacía y el bucle WHILE ... WEND se ejecuta de nuevo. En cuanto el usuario pulsa una tecla, el bucle se interrumpe y el programa continúa. Este mecanismo se suele programar al final de la zona en la que se escriben instrucciones en la pantalla; así se permite al usuario que se tome todo el tiempo que necesite para leer las instrucciones. Una vez dispuesto para continuar, el usuario pulsa una tecla para continuar la ejecución del programa. Como siempre, cuando decimos «cualquier tecla» damos por entendido que se exceptúan las teclas SHIFT, CTRL y CAPS LOCK, que no tienen ningún efecto, y la tecla ESC, que interrumpe el programa.

```
10 CLS
20 PRINT"PULSE CUALQUIER TECLA"
30 WHILE INKEY$="":WEND
40 PRINT"FIN"
```

Fig. 4.15 Utilización de INKEY$ en un bucle WHILE ... WEND para detectar cuándo se ha pulsado una tecla.

La instrucción INKEY$ genera una cadena literal si hay una tecla pulsada, de modo que podemos asignar INKEY$ a una variable literal, T$. Así, cada vez que se pulsa una tecla el carácter correspondiente es asignado a T$, y ésta cadena puede ser manipulada o examinada según convenga. En la figura 4.16 se ilustra cómo utilizar INKEY$ para captar una respuesta del tipo S/N; en este programa se incluye una trampa para errores. Se ha incluido INKEY$ en un bucle que comprueba incesantemente el teclado (línea 20). El bucle se interrumpe cuando T$ no es una cadena vacía. Entonces se comprueba T$ para determinar si es una respuesta aceptable. En un caso como éste resulta algo torpe utilizar un bucle WHILE ... WEND; es preferible esta versión más sencilla.

```
10 PRINT"POR FAVOR, TECLEE S O N"
20 T$=INKEY$:IF T$=""THEN 20
30 IF T$<>"N" AND T$<>"S" THEN PRINT"RESPUESTA
   INCORRECTA. TECLEE S O N"
40 PRINT"SU RESPUESTA HA SIDO";T$
```

Fig. 4.16 Captación de una respuesta «S» o «N» con INKEY$.

Hay otro método diferente de determinar si se está pulsando una tecla. La instrucción que lo realiza es INKEY y la idea básica es que la pulsación de una tecla produce un código numérico. Estos códigos numéricos son los que figuran en el manual, Apéndice III, página 16. Con INKEY no se realiza un mecanismo del tipo «esperar hasta que esté preparado»; sólo sive para detectar una tecla concreta. La figura 4.17 ilustra esta función, con INKEY(47) para detectar si se está pulsando la barra espaciadora. INKEY(47) = 0 prueba si la barra espaciadora está pulsada, y INKEY(47) = −1 si no lo está. Hay otros números para determinar si las teclas SHIFT y CTRL (o ambas simultáneamente) están siendo pulsadas al mismo tiempo que la que se desea examinar. Por ejemplo, INKEY(47) = 32 comprueba si la barra espaciadora y SHIT están siendo pulsadas al mismo tiempo. INKEY(47) = 128 comprueba la barra espaciadora y CTRL. INKEY(47) = 160 comprueba las tres teclas al mismo tiempo.

INKEY es una forma particularmente útil de examinar teclas, especialmente en la programación de juegos, ya que permite comprobar todas las teclas, incluso las de movimiento del cursor, y las teclas COPY, DEL, ESC, etc.

```
10 PRINT"PULSE BARRA ESPACIADORA PARA CONTINUAR"
20 WHILE INKEY(47)<>"X"0:WEND
30 PRINT"CONTINUE AQUI ..."
```

Fig. 4.17 Utilización de la instrucción INKEY para detectar la pulsación de una tecla concreta.

5

Cadenas literales

FUNCIONES LITERALES

En el capítulo 3 hemos estudiado, aunque brevemente, unas cuantas funciones numéricas. Si al lector le gustan los números, tanto mejor. Pero las funciones literales pueden ser bastante más interesantes. Lo que las hace tan atractivas es el hecho de que las acciones más espectaculares de que es capaz el ordenador se realizan frecuentemente por medio de *funciones literales*. Empecemos por decir qué es una función literal. Para nuestros propósitos, función literal es toda acción que puede ser efectuada con cadenas literales. Por si esta definición no deja las cosas claras, entremos en detalle.

Para el CPC464 una cadena literal es un conjunto cualquiera de caracteres. Las cadenas pueden ser representadas por *variables literales*, cuyo nombre, como ya sabemos, tiene que terminar en un signo de dólar ($). En el CPC464 la longitud máxima de las cadenas es de 255 caracteres, y esto es más de lo que normalmente se puede necesitar. Al igual que muchos otros ordenadores, el CPC464 almacena las cadenas de una forma especial, utilizando lo que llamamos código ASCII. Estas letras son las siglas de «American Standard Code for Information Interchange» (código americano estándar para el intercambio de información). El código ASCII es el utilizado por la mayor parte de los ordenadores. No se debe confundir este código por el utilizado en la función INKEY. En la figura 5.1 se muestra un listado de los códigos ASCII junto con los caracteres correspondientes.

Cada carácter está representado por un número, del 32 (que es el espacio) al 127. En mi impresora el carácter 127 es un cuadrado negro, pero en la pantalla del CPC464 es un mosaico formado por pequeños rectángulos blancos y negros. Se pueden asignar caracteres a una variable literal utilizando el signo =. Al hacer la asignación de esta manera, los caracteres tienen que estar encerrados entre comillas. Cuando la asignación se realiza por medio de instrucciones INPUT o READ ... DATA las comillas no son necesarias.

Como sabemos, hay diferencias radicales entre la forma de almacenar en la memoria los números y las cadenas literales. Puesto que una cadena consiste en una sucesión de códigos almacenados en la memoria del ordenador, podemos hacer con las cadenas manipulaciones que no son posibles con los números. Por ejemplo, podemos determinar muy fácilmente cuántos caracteres tiene una cadena. Podemos

32		33	!	34	"
35	#	36	$	37	%
38	&	39	'	40	(
41	)	42	*	43	+
44	,	45	–	46	.
47	/	48	0	49	1
50	2	51	3	52	4
53	5	54	6	55	7
56	8	57	9	58	:
59	;	60	⟨	61	=
62	⟩	63	?	64	@
65	A	66	B	67	C
68	D	69	E	70	F
71	G	72	H	73	I
74	J	75	K	76	L
77	M	78	N	79	O
80	P	81	Q	82	R
83	S	84	T	85	U
86	V	87	W	88	X
89	Y	90	Z	91	[
92	\	93	]	94	↑
95	_	96	'	97	a
98	b	99	c	100	d
101	e	102	f	103	g
104	h	105	i	106	j
107	k	108	l	109	m
110	n	111	o	112	p
113	q	114	r	115	s
116	t	117	u	118	v
119	w	120	x	121	y
122	z	123	{	124	¦
125	}	126	~	127	

Fig. 5.1 Caracteres del código ASCII.

extraer determinados caracteres de ella, o también cambiar alguno o insertar otros nuevos. Las acciones de este tipo son lo que denominamos funciones literales.

Hemos mencionado que una de las funciones literales posibles es determinar el número de caracteres de que consta una cadena. Puesto que las cadenas pueden

llegar a tener hasta 255 caracteres, es muy conveniente disponer de un método automático para contarlos. El método lo proporciona la función LEN. La palabra LEN debe ir seguida del nombre de la variable literal, entre paréntesis. El resultado es siempre un número, que en consecuencia podemos manejar como tal; por ejemplo, escribiéndolo en la pantalla o asignándolo a alguna variable numérica. No es necesario dejar espacio entre la N de LEN y el signo de abrir paréntesis.

```
10 CLS
20 A$="AL CPC464 SE LE PUEDEN CONECTAR DIVERSOS
   PERIFERICOS":PRINT A$
30 PRINT"EN ESTA FRASE HAY";LEN(A$);"CARACTERES."
40 INPUT"PRUEBE USTED CON OTRA FRASE: ";B$
50 PRINT B$;"TIENE";LEN(B$);"CARACTERES."
```

Fig. 5.2 Presentación de LEN, primer miembro de la familia de funciones literales.

La figura 5.2 muestra un ejemplo de utilización de LEN. La línea 20 asigna una cadena a una variable literal; la línea 30 escriba el número de caracteres de que consta esa cadena (es decir, su longitud). Observe que decimos «caracteres», no «letras». Tanto los diversos signos como el espacio cuentan como caracteres, ya que todos ellos están representados por algún código ASCII. Las líneas 40 y 50 calculan y escriben la longitud de la cadena tecleada por el usuario. Esto puede ser útil para asegurarse de que un nombre introducido por el teclado no es excesivamente largo para una aplicación concreta. Por ejemplo, podría tratarse de un programa que escriba nombres en un formulario, con columnas de anchura limitada. Si se escribiera un nombre demasiado largo, parte de él aparecería en la columna correcta y el resto en la siguiente. Quizá haya observado, si su nombre o dirección son suficientemente largos, que cuando recibe cartas con la etiqueta impresa por ordenador parte del nombre o de las señas está abreviado. Ahora ya sabe por qué.

```
10 CLS
20 T$="EL ASOMBROSO AMSTRAD"
30 PRINT TAB(42-LEN(T$))/2);T$
```

Fig. 5.3 Centrado de títulos mediante LEN.

Esto no parece demasiado impresionante, pero podemos sacarle provecho, como demuestra el programa de la figura 5.3. En este programa se utiliza LEN como parte de una rutina que escribe una cadena llamada T$ centrada en una línea. Esta rutina es verdaderamente útil, y el lector tendrá ocasión de utilizarla repetidamente en sus propios programas porque le ahorrará el tedioso esfuerzo de contar los caracteres

del texto que quiera escribir centrado. LEN calcula la longitud de la cadena T$. Este número se resta de 42, y el resultado se divide por 2. Si la longitud de la cadena es impar, el número de TAB contendrá un 0.5, pero esta parte decimal será ignorada en el momento de escribir. Dicho sea de paso, se puede utilizar el número 41 o el 42. En cualquier caso, los textos quedan razonablemente bien centrados. El 41 funciona mejor cuando la longitud de la cadena es impar; el 42 cuando la longitud es par. Como el lector ya estará pensando, se puede afinar más y elegir 41 o 42, según convenga, utilizando algún IF ... THEN ... ELSE. Una rutina así programada se puede aplicar entonces a cualquier cadena cuyo nombre sea T$. En el próximo capítulo estudiaremos las *subrutinas*; son zonas de un programa que se teclean una sola vez y pueden ser utilizadas repetidamente en determinadas condiciones.

STR$ Y VAL

Ya hemos indicado que hay operaciones que pueden ser realizadas con números y no con cadenas, y que hay otras que operan sobre cadenas y no sobre números. En algún caso esto puede ser inconveniente, pero siempre podemos convertir los dígitos de una forma a otra, y muy fácilmente. Esta conversión permite efectuar operaciones aritméticas con números que tengamos en forma de cadena, y aplicar funciones literales a números que tengamos en forma de variable numérica. Consideremos el programa de la figura 5.4. Para empezar, definimos N$ como número en forma

```
10 N$="22.5":V=2
20 CLS:PRINT
30 PRINT N$;" POR";V;"ES";V*VAL(N$)
40 PRINT
50 V$=STR$(V)
60 PRINT"HAY";LEN(V$);"CARACTERES EN";V"!"
70 PRINT
80 PRINT N$;" MAS ";V$;" ES ";N$+V$;" ?"
```

Fig. 5.4 Conversión de forma numérica a literal, y viceversa, utilizando STR$ y VAL. Nótese el espacio introducido por STR$.

literal, y V como número en forma numérica. La línea 30 ilustra cómo se pueden realizar cálculos aritméticos con números que estén en forma literal. Basta con poner VAL(N$) en lugar de N$ para que lo que se utilice en la operación sea el valor numérico de N$. En la línea 50 transformamos V en su forma literal, V$. Pero en la línea 60 podemos observar lo siguiente: V era el número 2, es decir, un número de un solo dígito; sin embargo, al hacer la conversión con STR$, el número de caracteres ha aumentado misteriosamente a dos. Esto se debe al espacio que el ordenador reserva para un posible signo + o −. La función STR$ siempre introduce este espacio (me-

jor dicho, lo mantiene). Así pues, una cadena que haya sido obtenida a partir de un número siempre tendrá un carácter de más, salvo que el número original lleve el signo –. La línea 80 demuestra qué ocurre si intenta sumar dos cadenas sin previamente convertirlas a forma numérica con VAL.

DISECCIÓN DE CADENAS

El siguiente grupo de funciones literales está formado por lo que denominamos *operaciones de disección*. El resultado de cortar una cadena es otra cadena: un trozo tomado de la cadena original. La disección permite averiguar qué caracteres están en diversos lugares de una cadena.

Esto parece irresistiblemente interesante, así que vamos a ver enseguida un ejemplo, figura 5.5. En la línea 20 se asigna una cadena a la variable A$, y en la línea

```
10 CLS
20 A$="AMSTRAD"
30 B$="IGO PARA SIEMPRE"
40 PRINT LEFT$(A$,2)+B$
```

Fig. 5.5 Disección de una cadena con la función LEFT$.

30 otra cadena a B$. En la línea 40 se escribe una frase que utiliza las dos primeras letras de la palabra «AMSTRAD». ¿Cómo ha ocurrido esto? La función LEFT$ significa «copiar parte de la cadena empezando por la izquierda». LEFT$ ha de ir seguido de dos parámetros, entre paréntesis y separados por una coma. El primero es el nombre de la variable de la que se va a extraer un trozo; el segundo es el número de caracteres que se van a extraer. El efecto de LEFT$(A$,2) es, pues, extraer los dos primeros caracteres de la palabra AMSTRAD; el resultado es AM. A estos dos caracteres se une la cadena definida en la línea 30. La cadena final se imprime en 40.

En la figura 5.6 se da una aplicación un poco más profesional. El programa extrae las iniciales de un nombre; para ello mezcla la función LEFT$ con la concatenación.

```
10 CLS:PRINT:PRINT
20 INPUT"POR FAVOR, SU NOMBRE DE PILA: ";NP$
30 INPUT"AHORA SU APELLIDO: ";AP$
40 PRINT:PRINT
50 PRINT"SI ME PERMITE, YO LE VOY A LLAMAR ";
   LEFT$(NP$,1);"."+LEFT$(AP$,1)+"."
```

Fig. 5.6 Extracción de las iniciales de un nombre con la función LEFT$.

Las instrucciones INPUT de las líneas 20 y 30 captan el nombre y el apellido y los asignan a las variables NP$ y AP$. La línea 50 escribe las iniciales extrayéndolas con la función LEFT$ y mezclándolas con los puntos. Una posible aplicación de esta idea está en los juegos: si hay dos jugadores, a veces es preferible visualizar junto al tanteo solamente las iniciales; los nombres completos se pueden guardar en la memoria para utilizarlos en algún otro lugar del programa.

DISECCIÓN POR LA DERECHA

La disección de cadenas no se limita a copiar unos cuantos caracteres tomados de su izquierda. También podemos extraer caracteres de la derecha de una cadena. Esta función no se utiliza tanto como LEFT$, pero no deja de tener su utilidad. En la figura 5.7 se da una idea de cómo evitar el tener que repetir la mecanografía de palabras. Por supuesto, la aplicación práctica es algo más seria. Por ejemplo, se pueden

```
10 CLS
20 A$="AMSTRAD MAGICO"
30 PRINT:PRINT
40 PRINT"DEMASIADO";RIGHT$(A$,6);"PARA MI"
```

Fig. 5.7 Extracción de caracteres de la derecha de una cadena literal mediante la función RIGHT$.

extraer los cuatro últimos dígitos de una cadena tal como 91-746-4276. Y decimos *cadena* deliberadamente, porque un dato como éste tiene que ser tratado como cadena, no como número. En efecto, si tratamos de asignar este dato a una variable numérica, el ordenador no entenderá lo que realmente tratamos de hacer; es decir, si escribimos N = 91-746-4276, el ordenador supondrá que queremos restar 746 de 91, y 4276 del resultado. Evitemos problemas escribiendo N$ = "91-746-4276".

Podemos conseguir efectos llamativos combinando LEFT$ con RIGHT$. El programa de la figura 5.8 hará diabluras con las letras de su nombre. En la línea 20 se capta el nombre, y se lo asigna a la variable A$. En la línea se asigna a la variable

```
10 CLS
20 INPUT"POR FAVOR, ESCRIBA SU NOMBRE: ";A$
30 N=LEN(A$)
40 FOR J=1 TO N
50 PRINT LEFT$(A$,J);TAB(21)RIGHT$(A$,J)
60 NEXT
```

Fig. 5.8 Disección por ambos lados de una cadena.

N el número de caracteres del nombre, incluidos espacios y guiones. En la línea 40 empieza un bucle cuyo límite superior es el número de caracteres. La línea 50 es la fundamental. Cuando J es 1, esta línea escribe la primera letra de la izquierda y la primera de la derecha. En la siguiente pasada del bucle se escriben dos letras de cada lado en la línea siguiente. Así se continúa hasta escribir el nombre completo. Si se trata de extraer con LEFT$ o RIGHT$ un número de caracteres mayor que la longitud de la cadena, lo que se obtiene es la cadena completa.

DISECCIÓN POR EL CENTRO

Todavía hay otra instrucción de disección de cadenas, mucho más potente que LEFT$ y RIGHT$. Se trata de la instrucción MID$, que debe ir seguida de tres parámetros, entre paréntesis y separados por comas. El primero es el nombre de la variable literal. El segundo es un número que indica en qué lugar comienza la disección, empezando a contar los caracteres por la izquierda. El tercer parámetro es el número de caracteres que se quiere extraer.

```
10 CLS
20 A$="AMSTRAD CPC464"
30 L=LEN(A$)
40 FOR N=1 TO L
50 PRINT MID$(A$,N,1);" ";:NEXT
60 PRINT:PRINT
70 FOR N=1 TO L
80 PRINT MID$(A$,N,1)+"+";:NEXT
```

Fig. 5.9 Utilización de la instrucción MID$. Esta instrucción puede extraer cualquier parte de una cadena, y puede ser controlada, lo mismo que LEFT$ y RIGHT$, por variables.

Para ver esta instrucción en acción consideremos el programa de la figura 5.9. La línea 20 asigna a A$ la cadena literal «AMSTRAD CPC464»; la línea 30 determina la longitud de esta cadena, L. El bucle empieza en la línea 40 y va leyendo, en cada pasada, una letra de A$. Para N = 1, la letra extraída es la A, porque su posición en la cadena es 1. Si hubiéramos escrito MID$(A$, 1,2) habríamos obtenido AM; con MID$(A$,3,2) habríamos obtenido ST. Tal como está el programa, extraemos una letra cada vez. El punto y coma de la línea 50 inhibe el salto a la línea siguiente cada vez que se escribe una letra. El efecto global es escribir las letras de A$ con espacios intercalados. El segundo bucle (líneas 70 y 80) actúa de forma similar, pero intercalando signos + en lugar de espacios.

Una de las características de las funciones de disección es que se las puede controlar con variables numéricas en lugar de números. El programa de la figura 5.10

realiza una disección algo más complicada. En la línea 20 capta una cadena. La línea 30 realiza unos cálculos; veamos para qué. Supongamos que el nombre tecleado es MANOLO. Esta cadena tiene 6 letras, luego la línea 30 asigna a L el valor 6; C es la parte entera de L/2 (que es 3) más 1, o sea, 4. En la línea 40 empieza un bucle de cuatro pasadas. En la primera se escribe en TAB(20) (ya que N = 1 y 21 − N es 20); los parámetros de MID$ son C − N + 1, que es 4 − 1 + 1 = 4, y N*2 − 1, que es 1. Así pues, lo que se escribe en la primera pasada es MID$(NM$,4,1), que es 0 en este ejemplo. En la segunda pasada del bucle N es 2, C − N + 1 es 3 y N*2 − 1 es tres. Lo que se escribe es MID$(NM$,3,3), o sea, NOL. El bucle continúa de esta forma; el resultado final es una pirámide de letras tomadas de su nombre. El efecto es tanto más impresionante cuanto más largo sea el nombre.

```
10 CLS
20 INPUT"POR FAVOR, TECLEE SU NOMBRE: ";NM$
30 L=LEN(NM$):C=INT(L/2)+1
40 FOR N=1 TO C
50 PRINT TAB(21-N)MID$(NM$,C-N+1,N*2-1)
60 NEXT
```

Fig. 5.10 Construcción de una pirámide de letras para ilustrar la función MID$ combinada con una fórmula.

OTROS CARACTERES

Continuemos estudiando más funciones literales. Si retrocede unas páginas, recordará que hemos hablado del código ASCII, que es el conjunto de números con los que se representan los caracteres que podemos escribir en la pantalla. Podemos averiguar qué código corresponde a cada letra utilizando la función ASC seguida de la letra en cuestión encerrada entre comillas y paréntesis, o bien del nombre de una variables literal entre paréntesis (pero sin comillas). El resultado de esta función es un número, el código ASCII correspondiente a la letra examinada. Si se aplica ASC a una cadena, el resultado es el código ASCII del primer carácter. Por ejemplo, ASC("CPC464") es el código ASCII de la letra C. Consideremos el ejemplo de la figura 5.11. La línea 10 asigna una cadena a la variable A$. La línea 30 inicia un

```
10 A$="AMSTRAD CPC464"
20 CLS:PRINT
30 FOR N=1 TO LEN(A$)
40 PRINT ASC(MID$(A$,N,1));" ";
50 NEXT
```

Fig. 5.11 Obtención del código ASCII de las letras mediante la función ASC.

bucle que recorre todas las letras de A$. Las letras son extraídas una a una mediante MID$(A$,N,1); la función ASC da el código ASCII de cada una de ellas. Observe cómo hemos utilizado los paréntesis. El espacio entre comillas y los puntos y comas de la línea 40 controlan la forma de escritura.

ASC tiene una función inversa, CHR$. El parámetro, entre paréntesis, de CHR$ tiene que ser un código ASCII, y el resultado es el carácter correspondiente a ese código. Por ejemplo, la instrucción PRINT CHR$(65) escribe en la pantalla la letra A, ya que 65 es el código ASCII de la letra A. Esta función permite la codificación de mensajes. A veces el programador desea ocultar un mensaje de forma que no sea evidente para quien lea el listado del programa. Los códigos ASCII no desorientarían a un experto, pero posiblemente sí al usuario inexperto. La figura 5.12 da un ejemplo de esto. La línea 40 contiene un bucle WHILE ... INKEY$ ... WEND para

```
10 CLS:PRINT
20 PRINT"CUAL ES LA CLAVE DEL EXITO EN INFORMATICA?"
30 PRINT"PULSE UNA TECLA PARA REVELAR EL SECRETO"
40 WHILE INKEY$="":WEND
50 FOR J%=1 TO 6
60 READ D%:PRINT CHR$(D%);
70 NEXT J%
100 DATA 67,80,67,52,54,52
```

Fig. 5.12 Utilización de los códigos ASCII para ocultar un mensaje en un programa y recuperación del mensaje con CHR$.

esperar por el usuario. Cuando éste pulsa una tecla, el bucle que empieza en 50 escribe seis letras en la pantalla. Los códigos ASCII de estas letras están en la lista DATA. La línea 60 los lee, los convierte en caracteres y los escribe (uno a continuación de otro, gracias al punto y coma). Para ocultar mejor los caracteres se podrían utilizar otros números no tan obvios; por ejemplo, la cuarta parte de los códigos ASCII, o los códigos más 20 multiplicados por 5, etc. Estos nuevos códigos se pueden guardar en la lista DATA y programar su correcta conversión a caracteres ASCII. Esto será suficiente para desanimar a todo curioso, salvo quizá a los más persistentes decodificadores. Dicho sea de paso, en este ejemplo se utiliza READ ... DATA en un bucle y se definen las variables J% y D% como enteras. Esto ahorra memoria y acelera la velocidad de ejecución del bucle.

Hay otra instrucción relacionada con READ y DATA. Se trata de RESTORE. Si la utilizamos aislada, esta instrucción «restaura» los datos haciendo que el puntero de DATA señale el primer dato. Por ejemplo, si se han leído seis datos de una lista DATA que contiene solamente seis elementos, ya no se puede ejecutar otra instrucción READ, porque no hay nada más que leer. En cambio, si se pone RESTORE inmediatamente antes de READ, el elemento que se lee es el primero de la lista. Otra forma de utilizar RESTORE es poner un número de línea a continuación;

entonces el primer elemento leído será el que figure en primer lugar en la línea especificada. Por supuesto, en esa línea tiene que haber una lista DATA. Veamos el programa de la figura 5.13. En función del número elegido por el usuario, las líneas 50, 60 y 70 ejecutan una instrucción RESTORE seguida de un número. Es lamentable que RESTORE sólo admita un número y no una variable numérica; si lo hiciera, podríamos hacer el programa más compacto poniendo, por ejemplo, RESTORE 1000 * N. Cada lista DATA contiene tres elementos y un finalizador X. El bucle que lee los datos está programado de tal forma que se interrumpe cuando lee X. El bucle también podría ser del tipo FOR ... NEXT, dado que el número de datos que hay que leer es siempre el mismo. No obstante, el método adoptado es válido también para los casos en los que las listas tengan diferente número de elementos y es, por lo tanto, más útil. RESTORE, utilizada de esta forma, proporciona una forma cómoda de seleccionar una lista cada vez que se ejecute un programa.

```
10 CLS
20 PRINT"QUE LISTA DEEA?"
30 INPUT"ELIJA (1/2/3):";N
40 IF N<1 OR N>3 THEN PRINT"RESPUESTA NO VALIDA.":
   GOTO 30
50 IF N=1 THEN RESTORE 140
60 IF N=2 THEN RESTORE 150
70 IF N=3 THEN RESTORE 160
80 A$=""
90 WHILE Q$<>"X"
100 PRINT Q$;" ";
110 READ Q$
120 WEND
130 END
140 DATA OPEL,MERCEDES,PORSCHE,X
150 DATA RENAULT,PEUGEOT,CITROEN,X
160 DATA FIAT,ALFA-ROMEO,LANCIA,X
```

Fig. 5.13 Utilización de RESTORE para seleccionar una lista de datos.

ORDENACIÓN DE CADENAS

En la figura 4.10 vimos cómo se pueden comparar los números. También es posible comparar cadenas literales utilizando los códigos ASCII como base para la comparación. Dos letras son iguales si tienen el mismo código ASCII; fácilmente se comprende, pues, qué puede significar el signo = aplicado a cadenas literales. Para que dos cadenas sean iguales, tienen que contener los mismos caracteres en el mismo

orden. El significado de < y > ya no es tan evidente. El código ASCII de la letra A es 65, y el de la letra B es 66. Teniendo esto en cuenta, se puede decir que A «es menor que» B porque su código ASCII es menor. Para colocar letras por orden alfabético basta entonces con disponerlas en orden de códigos ASCII crecientes.

Este proceso se puede generalizar para comparar palabras completas letra a letra.

```
10 CLS
20 A$="QWERTY"
30 PRINT:INPUT"ESCRIBA UNA PALABRA EN
   MAYUSCULAS: ";B$
40 IF A$=B$ THEN PRINT"IGUAL QUE LA MIA!":END
50 IF A$>B$ THEN Q$=A$:A$=B$:B$=Q$
60 PRINT"EL ORDEN CORRECTO ES ";A$;" Y LUEGO ";B$
70 END
```

Fig. 5.14 Comparación de palabras para ordenarlas alfabéticamente.

El programa de la figura 5.14 compara palabras utilizando los signos = y>. La línea 20 asigna a A$ una «palabra» compuesta por las primeras seis letras de una línea del teclado. La línea 30 capta una palabra por el teclado. Las líneas 40 y 50 realizan las compraciones. Si la palabra tecleada (asignada a B$) es idéntica a QWERTY, se escribe el mensaje de la línea 40 y el programa concluye. SI QWERTY es posterior en orden alfabético a la palabra introducida, entra en acción la línea 50. Por ejemplo, si se teclea PERIFERICO, puesto que Q es posterior a P en orden alfabético (y su código ASCII es mayor que el de P), la palabra B$ será menor que A$, y la línea 50 las intercambia. Esto se realiza asignando A$ a una nueva variable, Q$ (Q$ = «QWERTY»), luego asignando B$ a A$ (A$ = «PERIFERICO») y finalmente Q$ a B$ (B$ = «QWERTY»). La línea 60 escribe primero A$ y luego B$, que siempre estarán en orden alfabético. En efecto, si la palabra tecleada es posterior a QWERTY (por ejemplo, TERMINAL), entonces A$ no es «mayor que» B$ y la comprobación de la línea 50 da como resultado «falso». Las palabras no son intercambiadas y el orden A$ B$ es correcto. Nótese que el programa ordena correctamente palabras tales como QWERTZ y QUERTX; es decir, para el orden no cuenta sólo la primer letra.

LISTAS

Los nombres de variables que hemos manejado hasta ahora han sido útiles, pero su utilidad tiene una seria limitación. Supongamos, por ejemplo, que cierto programa requiere que se introduzca por el teclado una larga serie de datos numéricos. No sería nada práctico tener que asignar una variable a cada dato. El programa de la figura 5.15 ofrece una solución a este problema. Las líneas 10 a 40 generan un conjunto de diez puntuaciones. La variable de la línea 30 es nueva para nosotros. Es

```
10 CLS
20 FOR N=1 TO 10
30 A(N)=1+INT(RND(1)*100)
40 NEXT
50 PRINT
60 PRINT TAB(16)"LISTA DE PUNTUACIONES"
70 PRINT:FOR N=1 TO 10
80 PRINT TAB(2)"EL CANTANTE";N;"HA RECIBIDO";A(N);
   "PUNTOS"
90 NEXT
```

Fig. 5.15 Lista de variables numéricas indexadas. La idea es más sencilla de lo que su nombre puede sugerir.

una *variable indexada*; el *índice* es el número N. Esta terminología no es específica de la informática y viene siendo utilizada desde antes de que los ordenadores existieran. ¿Cuántas veces ha manejado usted una lista de artículos numerados con 1, 2, 3,...? Estos números son un índice, que sirve sencillamente para identificar los elementos. De la misma forma, utilizando las variables A(1), A(2), A(3),... podemos identificar elementos diferentes que tengan un nombre común A. Un miembro tal como A(2) tiene un nombre que se lee «A de dos».

La utilidad de este método reside en que permite utilizar un solo nombre para la lista completa y acceder a las variables individuales por su número de índice. Puesto que el índice puede ser también una variable numérica o una expresión, la flexibilidad es extraordinaria. En el programa de la figura 5.15 las líneas 20 a 40 generan los elementos de la lista. Cada elemento se obtiene generando un número aleatorio comprendido entre 0 y 100, y ese número se asigna a A(N). Las líneas 60 a 90 escriben en la pantalla la lista de las puntuaciones.

Vamos a hacer una prueba. Modifique el lector el programa de forma que el bucle sea FOR N = 1 TO 11 y luego ejecútelo. El ordenador emitirá el mensaje de error «Subscript out of range in 30» (índice fuera de margen en 30). La razón es que el ordenador, salvo que se le informe a tiempo, está preparado para manejar listas de hasta diez elementos, pero no más. Si el número de elementos ha de ser mayor que 10, hay que prepararlo adecuadamente; la preparación consiste en que el ordenador reserve la memoria necesaria para recibir los datos. La instrucción DIM (dimensionar) realiza esta reserva de memoria. En realidad, una instrucción tal como DIM(11) permite que la lista contenga 12 elementos, porque también podemos utilizar A(0) (pero no A(12)).

En resumen, dimensionar consiste en escribir, después de DIM, todos los nombres de las listas que se vayan a utilizar, poniendo después de cada nombre entre paréntesis el número *máximo de* elementos que vaya a tener cada lista. En el curso del programa no es necesario llegar a alcanzar ese número máximo, pero lo que no se puede hacer es sobrepasarlo. Si esto ocurriera, el programa se detendría con un mensaje

de error; entonces habría que modificar la instrucción DIM y volver a empezar, lo que no sería nada agradable si se estuviera tecleando una lista de cien números. En una misma instrucción DIM se puede dimensionar cuantas variables se desee, como muestra la figura 5.16. Aunque no sea necesario dimensionar listas que vayan a tener 10 elementos o menos, es buena costumbre hacerlo, ya que de esa manera se ahorra espacio de memoria.

```
10 CLS:DIM A(12),N$(12)
20 PRINT TAB(2)"INTRODUZCA NOMBRES Y PUNTUACIONES"
30 FOR N=1 TO 12
40 INPUT"NOMBRE ";N$(N)
50 INPUT"PUNTOS";A(N)
60 NEXT
70 CLS:TOTAL=0
80 PRINT TAB(16)"LISTA DE PUNTUACIONES":PRINT
90 FOR N=1 TO 12
100 PRINT TAB(2);N$(N);TAB(22);A(N)
110 TOTAL=TOTAL+A(N)
120 NEXT
130 PRINT
140 PRINT"LA MEDIA ES";TOTAL/12
```

Fig. 5.16 Listas numéricas y literales. En este caso han tenido que ser dimensionadas.

En la figura 5.16 se da otro ejemplo de manejo de listas. Este programa capta los nombres y puntuaciones de diez artículos. Cuando la lista está completa, se borra la pantalla y se iniciliza la variable «TOTAL» (línea 70). A continuación se escribe la lista; en cada pasada por el bucle se actualiza la suma total de puntos (línea 110), dato necesario para escribir la puntuación media al final. Lo que el lector debe observar en este ejemplo es que las listas pueden contener no sólo números, sino también cadenas literales. Un nombre más «matemático» para las listas es el de *vector*; en la figura 5.16 se utilizan vectores literales (nombres) y vectores numéricos (puntuaciones).

FILAS Y COLUMNAS

Ya conocemos los vectores o listas de elementos. Hay otra variedad de lista, denominada *matriz*. Una matriz es un vector de grupos de elementos, cada uno de los cuales es a su vez un vector. Podemos imaginar una matriz como un conjunto de filas y columnas; cada grupo sería una fila, y los elementos de un grupo determinado estarían en columnas diferentes. Aclaremos las ideas con el ejemplo de la figura 5.17.

```
10 CLS:DIM N$(3,2)
20 FOR N=1 TO 3
30 FOR J=1 TO 2
40 READ N$(N,J)
50 NEXT J:NEXT N
60 FOR N=1 TO 3
70 PRINT TAB(5);N$(N,1);TAB(20);N$(N,2)
80 NEXT
100 DATA CABALLO,POTRO,VACA,TERNERO,PERRO,
   CACHORRO
```

Fig. 5.17 Matriz de filas y columnas.

En este ejemplo utilizamos la variable N$ con dos índices. El primero es el número de fila; el segundo es el número de columna. Para leer los elementos de esta matriz necesitamos dos bucles FOR ... NEXT. La lectura se realiza en las líneas 20 a 50. El bucle de las líneas 60 a 80 escribe los elementos de la matriz en columnas. En este bucle N es el número de fila; los números de columna son 1 y 2. Las filas contienen nombres de animales distintos. Por columnas se separan los nombres que se dan a los animales adultos y pequeños, respectivamente. En este ejemplo hemos utilizado una *matriz literal*, pero también se pueden construir matrices numéricas. Es posible que el lector conozca las matrices numéricas y sus diferentes aplicaciones, en particular para la resolución de ecuaciones simultáneas.

El programa de la figura 5.18 es un poco más ambicioso. El objetivo es almacenar un conjunto de nombres y números de teléfono, que el programa capta por medio de las líneas 20 a 50. Una vez completa la matriz, se puede pedir el teléfono de una

```
10 CLS:DIM A$(50,2)
20 FOR N=1 TO 50
30 LOCATE 2,5:INPUT "NOMBRE ";A$(N,1)
40 LOCATE 2,7:INPUT "TEL. ";A$(N,2)
50 CLS:NEXT
60 CLS:LOCATE 14,1:PRINT"LSTA COMPLETA"
70 LOCATE 10,4:INPUT"ELIJA UNA INICIAL ";J$
80 FOR N=1 TO 50
90 IF J$=LEFT$(A$(N,1),1) THEN PRINT"NOMBRE
   ";A$(N,1);TAB(20)"TEL. ";A$(N,2)
100 NEXT
```

Fig. 5.18 Utilización de una matriz literal en una sencilla aplicación de listín telefónico.

persona sin más que teclear su inicial. Para no hacer el programa demasiado largo, no hemos incluido trampas para errores, siempre necesarias en estos casos. Por ejemplo, como mínimo habría que prever lo siguiente:

PRINT «Lo siento; no encuentro ningún nombre que empiece por»;J$

Por otra parte, como es de esperar que el usuario quiera hacer más de una consulta, habría que programar algún tipo de bucle para que, una vez encontrado un nombre, el programa vuelva a 70 a esperar otra inicial. Otra parte del programa tendría que encargarse de responder en caso de no encontrar ningún nombre con la inicial suministrada. Más adelante le daremos alguna pista acerca de estas cuestiones.

Una función muy útil y nada frecuente es la función literal INSTR. Permite determinar si una cadena literal está contenida en otra. La forma de utilizarla es

X = INSTR(A$,B$)

para averiguar si B$ está contenida en A$. Si lo está, X es la posición de A$ en la que se encuentra el primer carácter de B$. Si B$ no está contenida en A$, X es cero. Por supuesto, X será cero siempre que B$ sea más larga que A$. Si lo único que se desea es ver escrito el número, basta con teclear

PRINT INSTR(A$,B$)

El programa de la figura 5.19 es un ejemplo de utilización de la función INSTR.

```
10 CLS
20 A$="ALBERT HALL"
30 B$="ESTATUA DE LA LIBERTAD"
40 C$="TORRE DE PISA$
50 PRINT"EN A$, ERT ESTA EN LA POSICION";
   INSTR(A$,"ERT")
60 PRINT"EN B$, ERT ESTA EN LA POSICION";
   INSTR(B$,"ERT")
70 PRINT"EN C$, ERT ESTA EN LA POSICION";
   INSTR(C$,"ERT")
```

Fig. 5.19 Ejemplo de utilización de INSTR.

Las líneas 20 a 40 asignan nombres a las cadenas literales. Las líneas 50 a 70 hacen las comprobaciones y escriben los resultados. Obsérvese que las cadenas tienen que tener la forma exacta: para la función INSTR no es lo mismo Ert que ert o ERT. Otro ejemplo: supongamos que en un programa se ha definido la cadena A$ = = "SIsiCLAROclaroNATURALMENTEnaturalmentePOR SUPUESTOpor supuesto" y que se hace al usuario una pregunta del tipo sí/no. Se puede utilizar la instrucción INSTR para examinar la respuesta. Si el resultado de X = INSTR(A$, Respuesta$) es cero, lo más probable es que la respuesta no haya sido afirmativa.

6

Menúes, subrutinas y programas

En la figura 4.16 introdujimos la idea de hacer una selección basada en la tecla que pulse el usuario. En aquel ejemplo la posibilidad de elección era limitada: S o N. Ahora podemos generalizar esa idea a un tipo de rutina que denominamos *menú*. Un menú es una lista de posibilidades de elección, generalmente funciones de un programa. Eligiendo una de esas posibilidades el usuario decide qué parte del programa se debe ejecutar. Una forma de hacer la oferta es numerar las posibilidades y pedir al usuario que teclee el número correspondiente a la opción deseada. En la figura 6.1 se muestra cómo quedaría un menú típico en la pantalla. En un ordenador

```
                              MENU
1. Hacer una lista nueva de nombres.
2. Añadir nombres a la lista.
3. Leer nombres de la lista.
4. Borrar nombres de la lista.
5. Elegir un nombre.
6. Fin del programa.
                    POR FAVOR, ELIJA UN NUMERO
```

Fig. 6.1 Un menú típico.

provisto de un BASIC de la época de los visigodos tendríamos que utilizar una sucesión de instrucciones del tipo

```
IF K = 1 THEN 1000
IF K = 2 THEN 2000
etc.
```

Hay otros métodos más sencillos, basados en las instrucciones ON ...GOTO y ON ... GOSUB. Éstas representan dos formas diferentes de realizar la misma tarea, y las estudiaremos detenidamente porque no están incluidas en otras versiones de BASIC que el lector pueda haber utilizado anteriormente.

Para empezar supongamos que queremos elegir entre cuatro posibilidades tecleando un número del 1 al 4. Lo primero que tenemos que conseguir es que no se acepten números extraños a este margen, tales como 0, 5, − 10; es decir tenemos que programar una trampa para errores. En segundo lugar, la captación del número mediante INPUT es algo incómoda, porque obliga a pulsar ENTER después de teclear el número. Así pues, utilizaremos INKEY$ para captar la respuesta. Una vez comprobado que ésta es válida, utilizaremos el número captado para elegir un número de línea y hacer que la ejecución del programa continúe a partir de ella.

En el ejemplo de la figura 6.2 las líneas 10 a 60 escriben en la pantalla un título y un menú, utilizando instrucciones TAB para controlar el formato. A continuación

```
10 CLS:PRINT TAB(19)"MENU"
20 PRINT TAB(2)"1. PEDIDOS DEL DIA"
30 PRINT TAB(2)"2. SITUACION DEL ALMACEN"
40 PRINT TAB(2)"3. COMPRAS"
50 PRINT TAB(2)"4. RESUMEN"
60 PRINT:PRINT TAB(4)"ELIJA UN NUMERO DEL 1 AL 4"
70 K$=INKEY$:IF K$=""THEN 70 ELSE K=VAL(K$)
80 IF K>4 OR K<1 THEN PRINT"RESPUESTA INCORRECTA.
   TECLEE UN NUMERO DEL 1 AL 4":GOTO 70
90 ON K GOTO 110,130,150,170
100 END
110 PRINT"ESTA ES LA RUTINA 1"
120 GOTO 100
130 PRINT"ESTA ES LA RUTINA 2"
140 GOTO 100
150 PRINT"ESTA ES LA RUTINA 3"
160 GOTO 100
170 PRINT"ESTA ES LA RUTINA 4"
180 GOTO 100
```

Fig. 6.2 Un menú que utiliza la instrucción ON K GOTO.

se informa cómo realizar la elección (lo que puede ser obvio para el programador, pero quizá no para el usuario). La parte fundamental del programa empieza en la línea 70, en la que establece con INKEY$ un bucle que no se interrumpe mientras no se pulse alguna tecla. Cuando se pulsa una tecla, INKEY$ será una cadena literal, que se convierte en un número después de ELSE. En la línea 80 de comprueba si este número está dentro del margen permisible, de 1 a 4. Si no lo está, se escribe un mensaje que recuerda al usuario cuál es el margen correcto. Si la letra pulsada es literal, K = VAL(K$) da a K el valor 0, y este número es rechazado en la línea

80. No se debe nunca asignar el número 0 a una opción de un menú, porque entonces es bastante más difícil distinguir entre letras y números.

En la línea 90, con el valor K correcto, se realiza la bifurcación mediante la instrucción ON K GOTO 110,130,150,170. En esta lista de números el orden es esencial, pues de él despende el curso que siga el programa. Así, la rutina que empieza en la línea 110 es ejecutada si la opción elejida fue la 1, la línea 130 corresponde a la opción 2, etc. Por supuesto, los números de línea no tienen que estar en orden creciente, sino en el que imponga la situación de las rutinas. En este ejemplo cada rutina no hace más que escribir un mensaje en la pantalla y termina con GOTO 100. Esto hace que el programa termine al concluir la rutina. Si hubiéramos puesto GOTO 10, el programa retornaría al menú; esto es lo que frecuentemente se hace en la práctica, y en tal caso en el menú hay que ofrecer la opción «FIN» para que el programa pueda terminar sin necesidad de apagar la máquina.

PROGRAMACIÓN MODULAR

Muchos programas empiezan con un título, unas instrucciones para el usuario y un menú. Dependiendo de la elección que se haga en el menú, la acción se reconduce a alguna parte del programa y más tarde éste termina o vuelve al menú. El método de selección con ON K GOTO es cómodo, pero más útil es la programación de *subrutinas*. Una subrutina es una sección de un programa que se puede insertar y ejecutar en cualquier lugar de un programa más largo. Las subrutinas se invocan (o llaman) mediante la instrucción GOSUB seguida del número de línea en la que empieza la subrutina. Cuando la ejecución del programa llega a esta instrucción, salta a la línea especificada tras GOSUB, igual que si se tratara de una instrucción GOTO. Pero, a diferencia de GOTO, GOSUB tiene un retorno *automático*. Al final de las subrutinas se pone la instrucción RETURN (retorno); su efecto es hacer que la ejecución del programa continúe en la instrucción inmediatamente siguiente al GOSUB que invocó la subrutina. Consideremos el ejemplo de la figura 6.3. La línea 20 escribe un texto, y el punto y coma impide el salto a la siguiente línea de la pantalla.

```
10 CLS
20 PRINT"ESTO ES UNA ";
30 GOSUB 1000
40 PRINT" SENCILLA":PRINT:PRINT
50 PRINT"LAS ";:GOSUB 1000:PRINT" SON EL CAMINO QUE
   LLEVA A LA PROGRAMACION AVANZADA."
60 PRINT:PRINT:END
1000 PRINT"SUBRUTINA";
1010 RETURN
```

Fig. 6.3 Una subrutina sencilla.

La instrucción GOSUB 1000 de la línea 30 hace que se escriba en la pantalla la palabra «subrutina». El RETURN de la línea 1010 reconduce el programa a la línea 40, que es la instrucción inmediatamente posterior al GOSUB 1000 original. Análogamente ocurre cuando GOSUB forma parte de una línea de varias instrucciones, como muestra la línea 50. En ella, GOSUB 1000 hace que se escriba «subrutina» en la pantalla, pero esta vez el retorno envía el programa a la segunda instrucción PRINT de la línea 50, no a la línea 60.

```
10 CLS:PRINT
20 PRINT TAB(11)"QUE MONSTRUO PREFIERE?"
30 PRINT
40 PRINT TAB(2)"1. VAMPIRO"
50 PRINT TAB(2)"2. HOMBRE-LOBO"
60 PRINT TAB(2)"3. ZOMBIE"
70 PRINT TAB(2)"4. SARGENTO MAYOR"
80 PRINT TAB(2)"5. PATRULLA VOLANTE"
90 PRINT:PRINT TAB(2)"ELIJA UN NUMERO"
100 GOSUB 10000:REM RUTINA INKEY$
110 IF K<1 OR K>5 THEN PRINT"ELECCION NO VALIDA.
    SOLO DEL 1 AL 5.":PRINT"ELIJA OTRA VEZ:"
120 ON K GOSUB 1000,2000,3000,4000,5000
130 PRINT:PRINT"QUIERE JUGAR OTRA VEZ? (S/N)"
140 GOSUB 10000:IF K$="S"THEN 10
150 END
1000 PRINT"SANGRE, SANGRE, DULCE SANGRE!":RETURN
2000 PRINT"AUUUUUUUUH!":RETURN
3000 PRINT"ZZZZZZZZZ!:RETURN
4000 PRINT"AL SUELO, RECLUTA!":RETURN
5000 PRINT"PSHIU, PSHIU!":RETURN
10000 K$=INKEY$:IF K$=""THEN 10000 ELSE K=VAL(K$)
10010 RETURN
```

Fig. 6.4 Selección de subrutinas en un menú.

Pasemos ahora a algo más complicado. En la figura 6.4 se utilizan subrutinas como parte de un programa de juegos (imaginario). Las líneas 10 a 80 ofrecen los juegos posibles. La línea 90 solicita la elección. A continuación viene el bucle INKEY$ y la trampa para errores. La línea 120 realiza la selección. Cualquiera que sea la rutina elegida, el programa retorna al GOSUB siguiente al elegido. Por ejemplo, si se ha pulsado la tecla del 1, la rutina que se ejecuta es la que empieza en 1000; el pro-

grama retorna a 120 para determinar si también tiene que ejecutar las restantes (2000, 3000, 4000 o 5000). Como K sigue siendo 1, el programa pasa a la línea 130, para averiguar si el usuario quiere terminar el programa o volver al menú. Si en la línea 1000 se hubiera alterado el valor de K a 2, 3, 4 o 5, automáticamente se ejecutaría la rutina correspondiente a este nuevo valor. Si por alguna razón se necesitara utilizar la variable K en las subrutinas, todas ellas deberían terminar con K = 0 para evitar una segunda selección no deseada.

Las subrutinas son útiles como opciones de un menú, pero lo son aun más como secciones de un programa que deban ser ejecutadas repetidamente. Por ejemplo, en la figura 6.4 el bucle INKEY$ ha sido programado como subrutina; puesto que el programa lo utiliza al menos dos veces, hemos ahorrado tiempo de mecanografía y espacio de memoria. Si ampliáramos el programa, cada vez que necesitáramos ese bucle bastaría con incluir la instrucción GOSUB 1000. Nótese que las subrutinas de este programa están en lugares tales que sólo se las puede ejecutar invocándolas con GOSUB, pues la línea 150 impide que la ejecución del programa llegue normalmente a ellas. Si no se toma esta precaución y el programa llega a una subrutina sin haber sido requerido por un GOSUB, al encontrar la instrucción RETURN inesperadamente el ordenador emite el mensaje «Unexpected RETURN in ...» (RETURN inesperado en...) y el programa se detiene. Así pues, es aconsejable colocar las rutinas en una zona del programa que vaya precedida por una instrucción END o STOP.

```
10 CLS
20 PRINT"POR FAVOR, ELIJA 1 O 2 "
30 GOSUB 1000
40 PRINT"HA ELEJIDO ";K$
50 END
1000 K$=INKEY$
1010 IF K$<>""THEN RETURN
1020 PRINT"*";:GOSUB 2000
1030 PRINT CHR$(8);CHR$(16);:GOSUB 2000
1040 GOTO 1000
2000 FOR J=1 TO 200:NEXT:RETURN
```

Fig. 6.5 Rutina para hcer parpadear un asterisco hasta que el usuario pulse una tecla.

En la figura 6.5 se muestra una variante de la subrutina INKEY$. El defecto de la instrucción INKEY$ consiste en que el usuario no sabe que está siendo utilizada, pues el ordenador no imprime el signo de interrogación como para la instrucción INPUT. Las líneas 1000 to 1040 hacen que aparezca en la pantalla un asterisco parpadeante mientras el usuario toma su decisión. Se hace parpadear el asterisco escribiéndolo y borrándolo alternadamente. CHR$(8) hace retroceder el cursor; CHR$(16) borra el carácter que se encuentre en la posición del cursor. Para hacer el parpadeo visible (es decir, suficientemente lento), se ha incluido otra subrutina:

un retardo en la línea 2000. Pruebe estas subrutinas en sus programas y verá cómo resultan más agradables de usar. Mientras tanto, procure ir haciendo buenas migas con las subrutinas. No sólo sirven para tener acceso a acciones programadas previamente, sino que constituyen una ayuda indispensable en la planificación de programas.

PONGASE EN MARCHA

Para justificar la inversión en su nuevo CPC464, al lector quizá le baste con utilizar los programas que puede adquirir en el comercio. Más instructivo e interesante es que introduzca y pruebe los programas que se publican en las revistas especializadas, posiblemente modificándolos para adaptarlos a sus preferencias o a sus necesidades. Pero la máxima satisfacción la obtendrá, sin duda alguna, del diseño de sus propios programas. Éstos no tienen por qué ser obras maestras. Concebir un programa, escribirlo y comprobar que funciona ya es placer suficiente, sobre todo por el hecho de que es su propia obra. Después de todo, comprar un ordenador y no programarlo es como comprar un Ferrari para que lo conduzca un chófer.

Algunos lectores querrán escribir programas que pongan orden en su colección de sellos o de discos, o para archivar recetas de cocina o datos técnicos de locomotoras de vapor antiguas. Los programas de este tipo se llaman *bases de datos*, pues su misión principal es recopilar, ordenar y archivar datos. Otros lectores estarán más interesados en juegos, manejo de colores, sonidos, dibujos u otros programas basados en los gráficos y en su movimiento. Estos programas utilizan las técnicas que describiremos detalladamente en los capítulos 8 a 10. En esta sección vamos a describir las bases de datos, pues su diseño tiene utilidad en todos los programas. En cuanto el lector sea capaz de escribir programas de este tipo, estará preparado para progresar, utilizando los mismos métodos, hasta el diseño de programas gráficos y de sonido. Tenga en cuenta, no obstante, que el movimiento rápido de los gráficos no se puede programar en BASIC. La razón es que la lentitud de BASIC no permite dar realismo a los movimientos ni controlar numerosos objetos móviles. Los programas de juegos gráficos que puede adquirir en las tiendas están escritos en *código de máquina*, lo que significa que las instrucciones son un conjunto de números, que es lo que el microprocesador (corazón del ordenador) entiende directamente. La programación en código de máquina no utiliza para nada el BASIC, y es mucho más difícil. De hecho, muchos programas en código de máquina se elaboran con ayuda de otros programas en grandes ordenadores, y luego se los adapta para su utilización en ordenadores domésticos. No obstante, quien aprende a programar en BASIC llega a poder programar en código de máquina; sólo es cuestión de experiencia, muchísima experiencia.

Hagamos dos observaciones muy importantes. Primero, que la experiencia es decisiva en el diseño de programas. Si sus primeros esfuerzos los dirige a proyectos no ambiciosos, aprenderá mucho de ellos. En efecto, la probabilidad de tener éxito es tanto mayor cuanto más sencillo sea el programa. Se aprende más de un pequeño programa que funciona a la primera o tras unos pequeños retoques que de un gran programa que no se termina de depurar nunca. En segundo lugar, el diseño de un programa empieza por desconectar el ordenador, y preferiblemente a buena distan-

cia de él. La razón es que el diseño es en principio un trabajo de planificación, y ésta es incompatible con la tentación que representa tener el teclado al lado.

LAPIZ Y PAPEL

Empiece por preparar papel en abundancia. Yo lo utilizo de formato A4, y con agujeros que me permitan archivar las hojas. De esta forma las mantengo ordenadas y puedo intercalar otras donde sea necesario. Observe que hablo de hojas, en plural. El diseño de un programa, por sencillo que parezca en principio, requerirá más de una. Si es programa es medianamente complejo, es fácil que antes de llegar al teclado haya llenado un par de docenas de hojas con notas de diseño y listas de instrucciones. Vamos a tomar un ejemplo de programa y a desarrollar su diseño. El programa es muy sencillo, pero ilustrativo.

Empezaremos por escribir qué queremos que el programa haga. Esto puede parecer superfluo, porque siempre creemos tener las ideas claras, pero no lo es. Se dice que los árboles impiden ver el bosque, y el dicho es muy cierto aplicado a la informática. Si se empieza a escribir directamente en BASIC, se llega a estar tan absorto con los detalles que al final se acaba por olvidar para qué serive toda esa maraña de instrucciones. En cambio, si escribimos en un papel nuestro objetivo, tendremos una meta a la que llegar, y esto es tan importante en el diseño de programas como en la vida. Y no basta con escribir unas cuantas palabras; el problema tiene que ser desmenuzado en todo detalle, pues de no hacerlo así no podremos programarlo. Por otra parte, un análisis suficientemente detallado hace posible el diseño de un programa bien *estructurado*; esto significa que las diversas secciones del programa forman una sucesión lógica, fácilmente susceptible de ampliación y modificación. Si el lector aprende a programar con esta disciplina, sus programas serán fáciles de entender, tardarán menos en funcionar y podrán ser ampliados para que realicen más funciones que las previstas en las primeras etapas de su concepción.

Consideremos, como ejemplo, la figura 6.6. Esto es un esbozo de la programación de un juego educativo sencillo. El plan del programa indica qué se espera del

---

Objetivos
1. Escribir una palabra en castellano en la pantalla.
2. Preguntar cómo se escribe esa palabra en inglés.
3. La respuesta tiene que ser correcta, incluso en ortografía.
4. El usuario no debe poder leer las respuestas en un listado del programa.
5. Dar un punto por cada respuesta acertada.
6. Permitir un segundo intento en cada pregunta.
7. Llevar la cuenta del número de intentos.
8. Presentar como resultado final el número de aciertos y el de intentos.
9. Elegir las palabras propuestas aleatoriamente.

Fig. 6.6 Esbozo del plan de un programa (¡de su primer programa!).

juego. Tiene que escribir en la pantalla una palabra en castellano, tomada al azar de una lista, y preguntar como se escribe en inglés. Además, tiene que comprobar la ortografía de la respuesta. También hay que evitar que el usuario (posiblemente familiar del programador) averigüe las respuestas haciendo un listado del programa y buscando las líneas de DATA. Todo juego tiene que disponer de un sistema de tanteo; vamos a dar un punto por cada respuesta acertada. Puesto que la ortografía ha de ser correcta, quizá haya que dar más de una oportunidad en cada pregunta. Finalmente, hay que llevar la cuenta no sólo de los aciertos, sino también de los intentos, para informar al final de cada juego. Y esto es suficiente, a no ser que queramos complicar más el juego. Para ser un primer programa, no está nada mal. ¿Qué tenemos que hacer ahora?

La respuesta es la siguiente: tenemos que actuar de la misma manera que un pintor hace su cuadro o un arquitecto diseña una casa, es decir, ocupándonos primero de los esquemas y luego de los detalles. Los esquemas de este programa son las etapas que forman el conjunto de acciones. Es evidente que tendremos que definir variables, dimensionar listas y hacer otros preparativos similares. Después tenemos que desarrollar el juego. Luego tendremos que ocuparnos de los contadores de aciertos e intentos. En la figura 6.7 se indica qué es lo que deberíamos tener escrito en un papel a estas alturas.

---

1. Escribir en la pantalla el título y las instrucciones del juego.
2. Escribir en la pantalla la palabra propuesta.
3. Preguntar cómo se escribe esa palabra en inglés.
4. Captar la respuesta con INPUT.
5. Determinar (por comparación) si la respuesta es correcta.
6. Si lo es, sumar 1 al contador de puntos y preguntar si quiere otra pregunta.
7. Si es incorrecta, dar otra oportunidad.
8. Si la segunda respuesta también es incorrecta, hacer otra pregunta.
9. Terminar el programa cuando el usuario teclee N en respuesta a «¿Quiere otra pregunta?».

Fig. 6.7 Siguiente etapa del desarrollo del plan.

LOS CIMIENTOS

Ahora ya podemos escribir unas líneas de programa, si bien sólo las fundamentales. Lo que no podemos hacer de ningún modo en esta etapa es ponernos a llenar hojas y más hojas con líneas de BASIC. Como es bien sabido, los cimientos de toda obra son decisivos, y tienen que estar bien echados antes de empezar a levantar las paredes.

En la figura 6.8 se muestra la primera versión del programa. Consta de solamente catorce líneas, todo lo que necesitamos por ahora. Recuerde que esto son los *cimientos*, no el edificio. Además, como estamos en plena construcción, tenemos que prodigar carteles de «Precaución: obras!», es decir, de líneas que empiecen con REM.

```
10 CLS:GOSUB 1000
11 REM TITULO
20 GOSUB 1200
21 REM INSTRUCCIONES
30 GOSUB 1400
31 REM PREPARACION
40 GOSUB 2000
41 REM JUEGO
50 GOSUB 3000
51 REM TANTEO
60 GOSUB 4000
61 REM OTRO?
70 IF INSTR("Ss",K$)<>0 THEN 40
100 END
```

Fig. 6.8 Cimientos del programa ejemplo.

Esta palabra reservada es abreviatura de *remarks*, observaciones; cualquier línea de programa que comience con ella será ignorada por el ordenador. Esto significa que después de REM podemos escribir cualquier cosa, lógicamente comentarios y observaciones sobre el programa. Estas observaciones no aparecen en la pantalla cuando se ejecuta el programa, sino solamente cuando se hace un listado. En la figura 6.8 hemos puesto las líneas REM inmediatamente después de las líneas fundamentales. Más adelante, cuando el programa esté completo y comprobado y funcione perfectamente, las suprimiremos para ahorrar memoria y mejorar la velocidad de ejecución. Siempre conviene guardar una copia del programa con las líneas REM, y utilizar normalmente una copia «de trabajo» sin ellas. La primera sirve para el caso de que haya que introducir modificaciones; la segunda es más rápida y eficaz.

Pero volvamos al programa. Como se puede observar, consiste en un conjunto de instrucciones GOSUB, con llamadas a líneas que aún no hemos escrito. El programa sigue el esquema de la figura 6.7. La única línea que contiene un GOSUB es la 70. Esta línea comprueba la respuesta del usuario a la pregunta «¿Quiere seguir jugando?». Para captar esta respuesta tendremos que escribir una subrutina con INKEY$.

En la línea 70 el resultado de INSTR(«Ss»,K$) será 1 si el usuario responde con «S», y 2 si responde con «s». En cualquier otro caso, INSTR dará el valor 0, que se interpreta como si el usuario hubiera respondido con «no».

Estudie detenidamente estas catorce líneas porque son muy importantes. Las llamadas a todas esas subrutinas nos permiten comprobar fácilmente esta versión del programa; después de todo, no hay mucho en lo que podamos habernos equivocado. Ahora debemos decidir en qué orden vamos a escribir las subrutinas. El peor orden suele ser el orden en que aparecen en el programa. Las que manejan el título y las intrucciones se deben dejar para el final, pues son las menos importantes para

el desarrollo del programa. Si las escribimos al principio, siempre cabe la posibilidad de que más tarde introduzcamos en el programa mejoras que nos obliguen a escribirlas otra vez. En este momento una buena idea es escribir

9 GOTO 30

con lo que puenteamos todo lo referente a título e instrucciones. Esto sirve también para ahorrar tiempo cuando se está probando el programa.

El siguiente paso es ponernos al teclado (¡por fin!) e introducir el programa tal como lo tenemos. En las líneas en las que empiezan las subrutinas podemos poner instrucciones PRINT, como hicimos en el ejemplo de la figura 6.2. Así se puede comprobar que el programa funciona, al menos por ahora.

A continuación podemos grabar este núcleo del programa. Si lo tenemos grabado, podemos cargarlo para ir añadiéndole y comprobando subrutinas. En cada etapa, una vez comprobado el funcionamiento de lo que se haya añadido, lo volveremos a grabar, y así dispondremos siempre de la versión más moderna. Estas comprobaciones parciales son muy importantes. Si las aplazáramos para cuando el programa fuera demasiado largo, el trabajo de depuración llegaría a ser tedioso y quizá imposible. En cambio, si vamos comprobando cada rutina inmediatamente después de introducirla, siempre podremos «fiarnos» del trabajo que ya llevamos hecho y concentrarnos en los errores del último añadido.

LA ETAPA DE LAS SUBRUTINAS

Lo que nos queda por hacer, y no es poco, es escribir las subrutinas. Para algunas no tenemos que pensarlo mucho. Veamos, por ejemplo, la que tenemos que poner en la línea 4000. Se tratará del conocido bucle INKEY$, junto con algún PRINT. En la figura 6.9 se da una posible versión. Introdúzcala y compruebe el funcionamiento del programa resultante.

```
4000 PRINT"QUIERE JUGAR OTRA VEZ?"
4010 PRINT""POR FAVOR, RESPONDA CON S O N"
4020 K$=INKEY$:IF K$=""THEN 4020
4030 RETURN
```

Fig. 6.9 Subrutina para la línea 4000.

Vamos a abordar lo que el lector seguramente considerará más difícil: la subrutina que realiza el juego. Pues bien, para escribirla no tiene que aprender nada nuevo. Esta subrutina se realiza por el mismo procedimiento que el núcleo del programa; es decir, escribiendo detalladamente las funciones que queremos que realice y ordenando las etapas en las que se desarrollará la acción. Si encontramos algo para lo que no tengamos solución inmediata, podemos relegarlo a una nueva subrutina.

Consideremos la figura 6.10, en la que hemos elaborado un plan de la rutina de juego. El primer paso hemos tenido que pensarlo bien. Recordará el lector que no queríamos que el usuario pudiera ver las respuestas en las líneas DATA si se le ocurría listar el programa. La solución más sencilla es ocultarlas en forma de códigos ASCII; el jugador que quiera hacer trampa podrá descifrarlas, pero no sin trabajo. Hemos decidido poner las respuestas en líneas DATA dedicando tres dígitos a cada código. ¿Por qué tres dígitos? Porque, aunque para las letras mayúsculas basta con dos (por ejemplo, la «A» es el 65), para las minúsculas hacen falta tres (la «n» es el 110); todo lo que tenemos que hacer es poner 065 en lugar de 65, 066 en lugar de 66, etc.

1. Almacenar las respuestas en forma de códigos ASCII en una cadena literal.
2. Almacenar la lista de preguntas en una lista de cadenas.
3. El número con el que elige la palabra en castellano sirve para acceder también la respuesta.
4. Utilizar la variable IT para contar los inentos y la variable CT para contar los aciertos.
5. Utilizar la variable OP para contar el número de intentos en cada pregunta.

Fig. 6.10 Planificación de la rutina de juego.

Las palabras en castellano las vamos a guardar en una lista. Esto tiene varias ventajas. La primera es la facilidad con que podemos seleccionarlas aleatoriamente. Otra es que también facilita emparejar las preguntas con las respuestas. Si las preguntas son elementos de una lista, con índices del 1 al 10, podemos poner las respuestas en líneas DATA (un conjunto de código en cada una) y leerlas también en forma de lista.

También tenemos que decidir qué nombres vamos a dar a las variables. Es útil asignar nombres que estén relacionados en las magnitudes que representan. En este caso, IT para el número de intentos y CT para el contador de aciertos. La tercera variable, OP, representa en qué oportunidad, de las dos que se ofrecen con cada pregunta, nos encontramos. Finalmente, para la lista de las palabras propuestas elegimos P$.

En la figura 6.11 se da una subrutina que corresponde al plan de la figura 6.10. La acción consiste en generar un número aleatorio, utilizarlo para elegir y escribir

```
2000 OP=0:V=INT(10*RND(1))+1
2010 CLS:PRINT"LA PALABRA ES ";P$(V)
2020 PRINT:PRINT"EN INGLES SE DICE: ";
2030 INPUT X$:IT=IT+1
2040 GOSUB 5000
2041 REM HALLAR RESPUESTA CORRECTA
2050 RETURN
```

Fig. 6.11 Programación de la subrutina de juego.

una palabra y buscar la respuesta. Y eso es todo, porque la comprobación de la respuesta del usuario y el mantenimiento del contador se confían a otra subrutina. Siempre es conveniente desmenuzar las tareas al máximo, para no tener que escribir grandes trozos de programa de una vez.

La subrutina empieza en la línea 2000 «borrando» una variable. OP se iguala a cero para asegurar que esta variable tenga el valor correcto cada vez que se ejecute la subrutina. La segunda instrucción de esta línea genera un número aleatorio comprendido entre 1 y 10. La acción de las líneas 2010 a 2030 es evidente. Primero escribimos la palabra que corresponde al número aleatorio y luego preguntamos al usuario por la versión inglesa. La última instrucción de 2030 cuenta el número de intentos. Aquí es precisamente el sitio donde hay que poner este contador, pues debemos incrementar el número de intentos cada vez que se produzca una respuesta. Vamos a aplazar el momento la lectura de las líneas de datos, tarea que enconmendamos a la subrutina de la línea 5000. La línea 2041 nos recuerda cuál tendrá que ser el cometido de esa subrutina. La rutina de juego termina con su necesario RETURN.

ENTREMOS EN DETALLES

Después de grabar en cinta la subrutina de juego podemos empezar a pensar en los pequeños detalles. El primero puede ser uno que preceda o siga a la función de juego; por ejemplo, la rutina del contador de aciertos. Como ya sabemos, hay que empezar por planificarla (Fig. 6.12). Cada vez que el usuario dé una respueta correcta hay que incrementar la variable CT y volver al programa principal. En cambio, si

1. Si la respuesta es correcta, incrementar CT.
2. Si la respuesta es incorrecta y OP = 0, dar otra oportunidad y hacer OP = 1.
3. Si la segunda respuesta (OP = 1) también es incorrecta, pasar a la siguiente pregunta y hacer OP = 0.

Fig. 6.12 Planificación de la subrutina del contador de aciertos.

la respuesta es incorrecta, hay que escribir el mensaje y dar otra oportunidad. Si también la segunda respuesta es incorrecta, no hay más que hacer. Más adelante, cuando lleguemos al capítulo 11, podremos incluir sonidos en el programa; por ejemplo, un pitido corto para los errores y uno largo para los aciertos. Si le gusta la idea, inclúyala en el plan de la subrutina.

En la figura 6.13 se da la programación de este plan. La línea 3000 se encarga de las respuestas correctas. Hay que escribir un mensaje para informar al usuario; para no complicar esta línea, enviamos el programa a la línea 3200. La instrucción GOTO 3040 de la línea 3210 asegura que no se ejecute el resto de la subrutina cuando la respuesta haya sido correcta. En el caso de respuesta incorrecta entra en acción

```
3000 PRINT:IF X$=R$ THEN CT=CT+1:GOTO 3200
3010 IF OP=0 THEN GOTO 3300
3020 OP=0:PRINT"NO HA HABIDO SUERTE. OTRA VEZ SERA."
3030 FOR Q=1 TO 1000:NEXT
3040 RETURN
3200 PRINT"CORRECTO. SU PUNTUACION ES AHORA";CT
3210 PRINT"EN "IT;"INTENTOS.":GOSUB 7000:GOTO 3040
3300 PRINT"NO ES CORRECTO,"
3310 PRINT"PERO LE DOY OTRA OPORTUNIDAD":TR=TR-1
3320 GOSUB 7000:GO=1:GOSUB 2010:GOTO 3000
```

Fig. 6.13 Programación de la subrutina del contador de aciertos.

la línea 3010. Lo primero que hace es comprobrar el valor de OP. Si es cero, salta a la línea 3300 para escribir un mensaje con las instrucciones pertinentes. La línea 3320 invoca nuevamente la subrutina de 2010 para que el usuario pueda responder otra vez. La instrucción GOTO 3000 del final de la línea 3320 comprueba esta segunda respuesta.

Observe la astucia: la variable numérica OP empieza con valor cero. Si la respuesta es incorrecta y OP es 0, se ejecuta la línea 3010. Pero una de las funciones de la línea 3320 es hacer OP igual a 1. Cuando se introduce otra respuesta, ahora ya con OP = 1, la línea que entra en acción es la 3000; si esta segunda respuesta es también incorrecta, la línea 3010 no se puede ejecutar, ya que OP no es cero. El programa pasa, pues, a la línea siguiente, que es la 3020. Ésta pone OP a cero para la siguiente pregunta, escribe un mensaje, hace una pausa y deja que la subrutina termine con el RETURN de 3040.

```
1400 IT=0:CT=0:OP=0
1410 DIM Q$(10),R$(10)
1420 FOR J=1 TO 10:READ P$(J):NEXT
1430 FOR J=1 TO 10:READ R$(J):NEXT
1440 RETURN
```

Fig. 6.14 Subrutina que dimensiona las variables y construye las listas.

Ahora que estamos en el buen camino, podemos empezar a pulir el resto de las subrutinas. En la figura 6.14 se muestra la subrutina que se encarga de dimensionar las listas. La línea 1400 pone a cero todas las variables de los contadores. La línea 1410 dimensiona la lista P$, que utilizaremos para las palabras en castellano, y la lista R$, que representará las palabras en inglés. La línea 1420 lee los nombres de una lista de datos para asignarlos a P$; la línea 1430 lee los códigos para construir R$. Podemos dejar las líneas DATA para más tarde, como de costumbre.

Ahora llegamos al problema de buscar la respuesta correcta. Como lo hemos tenido en cuenta en la planificación, la solución no debería ser demasiado difícil. Su programación se muestra en la figura 6.15. La variable V contiene el número aleatorio; aquí la utilizaremos para seleccionar una de las cadenas de códigos ASCII, R$(V). Puesto que cada código consta de tres dígitos, debemos extraer tres caracteres de cada vez; ésta es la razón por la que ponemos STEP 3 en el bucle FOR ... NEXT de la línea 5000. En 5010 construimos la cadena respuesta, a la que asignamos el nombre R$. Recordemos que el ordenador no confunde la cadena R$ con la lista R$(V). A R$ se le asigna el valor de cadena vacía al principio de la línea 5000 para «borrar» la respueta anterior. La cadena R$ se construye a continuación leyendo tres dígitos cada vez, convirtiéndolos a forma numérica mediante VAL y

```
5000 R$="":FOR J=1 TO LEN(R$(V)) STEP 3
5010 R$=R$+CHR$(VAL(MID$(R$(V),J,3))):NEXT
5020 RETURN
```

Fig. 6.15 Comprobación de la respuesta.

luego a un carácter mediante CHR$. Este carácter se coloca al final de R$, y el proceso continúa hasta que no haya más dígitos que leer. Lo difícil ya está hecho. La figura 6.16 da la subrutina que se encarga de las instrucciones del juego, y la figura 6.17, la que escribe el título. Después de escribir el título, en MODE 0, se introduce un retardo. El modo 0 da letras fácilmente legibles y es adecuado a esta función, salvo que la frase no quepa en una línea de la pantalla. Finalmente, en la figura 6.18 se muestran las líneas de DATA y la subrutina de retardo.

```
1200 CLS:PRINT TAB(15)"INSTRUCCIONES"
1210 PRINT:PRINT TAB(4)"EL ORDENADOR LE PROPONDRA
     UNA PALABRA"
1220 PRINT"EN CASTELLANO. USTED DEBE ESCRIBIRLA EN"
1230 PRINT"INGLES, CUIDANDO LA ORTOGRAFIA Y"
1240 PRINT"PONIENDO LA PRIMERA LETRA MAYUSCULA."
1250 PRINT"EL ORDENADOR LLEVARA LA CUENTA DE LOS"
1260 PRINT"ACIERTOS. TENDRA DOS OPORTUNIDADES PARA"
1270 PRINT"CADA PREGUNTA."
1280 PRINT:PRINT"PULSE LA BARRA ESPACIADORA"
1290 PRINT"PARA EMPEZAR."
1300 IF INKEY(47)=-1 THEN 1300 ELSE RETURN
```

Fig. 6.16 Programación de las instrucciones. Siempre conviene dejar esta tarea para el final.

```
1000 MODE 0
1010 PRINT"VOCABULARIO INGLES"
1020 GOSUB 7000:MODE 1:RETURN
```

Fig. 6.17 Escritura del título.

```
6000 DATA PERRO,GATO,VACA,CABALLO,GALLINA,ZORRO,
     CANGURO,GANSO,LEON,CERDO
6001 DATA 068111103
6002 DATA 067097116
6003 DATA 067111119
6004 DATA 072111114115101
6005 DATA 072101110
6006 DATA 070111120
6007 DATA 075097110103907114111111
6008 DATA 071111111115101
6009 DATA 076105111110
6010 DATA 090105103
7000 FOR Q=1 TO 3000:NEXT:RETURN
```

Fig. 6.18 Líneas de datos.

Ahora podemos reunir todas estas subrutinas y probar el programa. Recuerde que el CPC464 permite grabar secciones del programa y luego mezclarlas con la orden MERGE. Puesto que el programa ha sido diseñado por módulos, será fácil modificarlo. Por ejemplo, se pueden modificar las líneas DATA, o poner más palabras (no olvide modificar entonces la instrucción DIM de 1410). Puede convertir este juego en cualquier otro que funcione a base de preguntas y respuestas sin más que cambiar los datos y las instrucciones. Puede incluir efectos sonoros espectaculares y gráficos que hagan el programa más atractivo. Uno de los defectos de este programa es que, tal como está, puede proponer la misma pregunta dos veces. Para subsanarlo, pruebe a hacer lo siguiente: intercambie la palabra elegida con la última y luego restrinja el margen de los números aleatorios. Por ejemplo, si el primer número generado es 5, intercambiamos la palabra número 5 con la número 10, y en la siguiente vuelta generamos un número aleatorio comprendido entre 1 y 9. Para ello, en lugar de la instrucción 10 * RND(1) + 1 tendremos que poner D * RND(1) + 1, donde D parte de un valor inicial igual a 10 y es decrementada cada vez que el usuario da una respuesta acertada.

Desde luego, es mucho lo que se puede hacer para convertir este programa en un juego verdaderamente interesante. Lo hemos utilizado como ejemplo para demostrar que el lector ya está en condiciones de empezar a programar. Considérelo como un «juego de bloques» que puede reconstruir en la forma que más le guste. Como

mínimo, le dará una idea de lo que podrá conseguir cuando conozca a fondo su CPC464. A medida que vaya creciendo su experiencia, podrá abordar programas más largos y complejos. Llegado el momento, querrá adquirir una impresora y una unidad de discos. Pero aun antes de eso, comprobará lo útil que es guardar copias de las subrutinas que vaya programando para luego incorporarlas a programas futuros con la orden MERGE.

Una observación final. Supongamos que ha escrito un programa que no funciona como esperaba. ¿Cómo empezar a corregirlo? Le haremos algunas sugerencias a este respecto en el apéndice A, dedicado a la depuración de programas y a su edición en la pantalla.

7

Archivo de datos en cinta

En CPC464, a diferencia de la mayor parte de los microordenadores, tiene incorporado un sistema de almacenamiento de programas y datos. Los otros ordenadores utilizan un magnetófono de cassette corriente, pero el del CPC464 ha sido diseñado específicamente para esta aplicación. Es de suponer que el lector, cualquiera que sea su experiencia anterior con ordenadores, no conozca este sistema de archivo de datos; por ello vamos a dedicar este capítulo al estudio de la grabación y lectura de datos en el CPC464. En el capítulo 1 ya hemos descrito el manejo del magnetófono en lo referente a la grabación y carga de programas. La terminología extraña, con palabras tales como *dispositivos, canales* y *tampones*, es la única dificultad que se puede atravesar en nuestro camino. En cuanto el lector asimile el significado de estos términos, la utilización del magnetófono empezará a parecerle sencilla e interesante, y podrá sacar mayor provecho de su ordenador. Expliquemos, pues, el significado de esas palabras.

Un *dispositivo* es, este contexto, un aparato que recibe o entrega datos. El teclado, por ejemplo, es un dispositivo, pues envía señales eléctricas al ordenador cada vez que se pulsa una tecla. La pantalla es otro dispositivo porque recibe del ordenador señales eléctricas que convierte en imágenes. El teclado es un dispositivo emisor: envía señales. La pantalla, en cambio, es un dispositivo receptor: recibe señales. En el caso concreto del CPC464, la pantalla se comporta como si fuera varios dispositivos integrados en uno solo, pues se la puede dividir en áreas a las que se accede independientemente. En el capítulo 8 veremos cómo hacerlo.

Hay otros dispositivos que pueden operar como receptores y como transmisores. Una *consola*, que es la combinación del teclado y la pantalla, transmite y recibe datos. También el magnetófono es un dispositivo bidireccional, lo mismo que las unidades de disco.

Hemos dicho que las señales eléctricas pasan de un dispositivo a otro y, efectivamente, esa transferencia de señales es una de las misiones del ordenador. Pero resulta mucho más útil estudiar, no las señales en sí, sino lo que ellas representan. Cada conjunto de señales representa una unidad de datos denominada *byte*; los *datos* son la materia prima con la que operan los ordenadores. Un byte es el dato mínimo necesario para contener un carácter ASCII o una orden de BASIC. Los datos pueden ser números o letras: cualquier cosa que el ordenador sea capaz de manejar. Si usted ha escrito un programa que almacena y ordena los nombres y los cumpleaños de

sus amigos por orden de fechas, para que el programa funcione habrá que suministrarle datos. En este ejemplo los datos serán los conjuntos de números y fechas de nacimiento. Si el programa está diseñado para archivar recetas de cocina, los datos serán las instrucciones de las recetas, los nombres de los platos y las cantidades los ingredientes. Todo ordenador del que se pretenda que sirva para algo más que para juegos sencillos tiene que se capaz de grabar y leer datos de este tipo, independientemente del programa que los genera o los utiliza.

Tal capacidad es imprescindible. El CPC464 utiliza su memoria para diversos propósitos, muchos de los cuales no están sometidos al control del usuario. Un programa muy largo que capte los datos y los utilice puede no caber en lamemoria. Mucho más práctico es disponer de un programa corto que reúna los datos con instrucciones INPUT y los grabe en cinta. De esta forma los datos estarán a salvo de cualquier accidente que pueda corromper la memoria del ordenador. Después tendrá que haber otro programa que manipule esos mismos datos. La cantidad de información que puede residir en la memoria en un momento dado es mucho mayor si los dos programas son independientes que si están reunidos en uno solo.

Pero, ¿qué tiene esto que ver con tampones, canales y dispositivos? Como ya sabemos, los dispositivos son los elementos del sistema que reciben y emiten datos. Los *canales* son las vías de transmisión. Meditemos un poco sobre el funcionamiento del CPC464: cuando usted pulsa una tecla, algo aparece en la pantalla; el teclado es un dispositivo, la pantalla es otro, y hay un canal que los une. Dicho de otro modo, hay un soporte físico por el que se transmiten señales eléctricas del teclado a la pantalla. Lo que importa al programador es que esos canales pueden ser controlados, en el sentido de que podemos abrir y cerrar canales a nuestro antojo. Por supuesto, no sería lógico cerrar todos los canales, salvo en aplicaciones muy especiales. Lo habitual es que se mantenga abierto el canal que une el teclado con la pantalla; sin embargo, si un programa requiere que el usuario teclee una palabra clave sin permitir que otra persona la vea en la pantalla, sería necesario cerrar ese canal. La figura 7.1 indica cómo hacerlo. La línea 10 solicita del usuario una palabra clave de cuatro letras. En la línea 20 la instrucción PRINT CHR$(21) cierra el canal teclado-pantalla. La línea 30 capta la palabra clave sin visualizarla en la pantalla. La línea 40 vuelve a abrir el canal para permitir la escritura de los siguientes mensajes. Como de costumbre, la línea 50 comprueba la palabra introducida; si no consta

```
10 PRINT"ESCRIBA LA PALABRA CLAVE DE 4 LETRAS"
20 PRINT CHR$(21)
30 INPUT B$
40 PRINT CHR$(6)
50 IF LEN(B$)<>4 THEN PRINT"CLAVE INCORRECTA.
   INTENTELO OTRA VEZ.":GOTO 10
60 PRINT"GRACIAS. CLAVE RECONOCIDA."
70 PRINT"TECLEE PRINT B$ PARA VERLA."
```

Fig. 7.1 Cómo desconectar el canal teclado-pantalla.

de cuatro letras, el programa vuelve a solicitarla. Si quiere comprobar que efectivamente el ordenador ha captado algo, teclee PRINT B$.

Como el lector ya sospecha, hay canales que se abren en el momento de encender el ordenador. Esto es evidente en el caso del canal teclado-pantalla. Este canal tiene normalmente el número 0. La estructura general de la instrucción PRINT es PRINT # x; como ya hemos tenido ocasión de comprobar, cuando el canal en el que escribe es el 0 basta con la versión resumida, PRINT. El CPC464 está diseñado para permitir el manejo de diez canales. De ellos hay tres (el 0, el 8 y el 9) que tienen misiones específicas y es conveniente no tratar de modificarlas. Los canales se identifican por su número, escribiendo antes el signo estadounidense de «número», #.

Como suele ocurrir en informática, los canales no están numerados del 1 al 10, sino del 0 al 9. Normalmente el canal #0 controla la pantalla, el #8 la impresora y el #9 el magnetófono. Quedan, pues, siete canales libres; en este capítulo y el siguiente vamos a describir algunas maneras de utilizarlos. El presente capítulo se dedica principalmente a explicar cómo se graban datos en cinta. Para ello es necesario saber dos cosas: primero, cómo seleccionar un *tampón* (*buffer*) y, segundo, cómo asociarlo al magnetófono. Un *tampón* es un almacén temporal para la transmisión de datos.

La forma de seleccionar un tampón y asignarlo al magnetófono consiste en utilizar una de las instrucciones OPEN: OPENIN y OPENOUT. Esta última, al igual que SAVE, tiene que ir seguida del nombre de un fichero; al ejecutarla, el magnetófono queda conectado al ordenador en espera de recibir datos. Esto no es más que una preparación, sin transferencia alguna de datos. También OPENIN tiene que ir seguida del nombre de un fichero, lo mismo que LOAD. Gracias al nombre del fichero, el ordenador save cuál seleccionar y lee la cinta hasta que encuentre ese nombre en ella.

TÉCNICAS DE ARCHIVO DE DATOS

¿Qué es un fichero?

Los términos *fichero* y *archivo* aparecen repetidamente en este libro. Un fichero puede ser un conjunto de caracteres lógicamente interrelacionados. Por ejemplo, los caracteres de que consta un programa en BASIC constituyen un fichero, ya que la relación entre ellos es tal que el programa no funciona si falta alguno. Un conjunto de nombres y direcciones en código ASCII es un fichero, porque forma un bloque de información relativa a un ente homogéneo; por ejemplo, al conjunto de nuestros amigos, clientes o acreedores. Otros ejemplos de ficheros son un conjunto de bytes en código de máquina y el conjunto de datos numéricos que utiliza un programa de análisis financiero. No obstante, en este libro consideraremos que un fichero es un conjunto de datos cualesquiera que podemos grabar en una cinta separadamente del programa que los va a utilizar. Así, un programa de contabilidad doméstica requerirá un fichero de conceptos y cantidades de dinero. Este fichero será generado y actualizado por un programa, y habrá que archivarlos para poder ponerlos a disposición del programa en la próxima ejecución. Por poner otro ejemplo, suponga-

	FICHERO DE AMIGOS
REGISTRO 1	
CAMPO 1	Nombre 1
CAMPO 2	Dirección 1
CAMPO 3	N.° de teléfono 1
CAMPO 4	Cumpleaños 1
REGISTRO 2	
CAMPO 1	Nombre 2
CAMPO 2	Dirección 2
CAMPO 3	N.° de teléfono 2
CAMPO 4	Cumpleaños 2
REGISTRO 3	
etc.	

Fig. 7.2 Significado de los términos *registro* y *campo.*

mos que usted ha diseñado un programa para ordenar y controlar su colección de discos de 78 rpm. El programa requerirá, sobre todo al principio, la introducción de gran cantidad de datos acerca de las grabaciones. Esta información constituye un fichero y en algún punto del programa tendrá que grabarla en cinta. ¿Por qué? Pues por la sencilla razón de que, para que el programa sea verdaderamente útil, tendrá que ser capaz de manejar una cantidad mayor de información que la cabe en la memoria del ordenador. Además, no parece práctico tener que modificar el programa cada vez que se añade un disco a la colección. De esto es de lo que trata este capítulo: de grabar la información con la que trabaja un programa; dicho de otra forma, de *archivar* la información.

No podemos hablar de métodos de archivo sin antes conocer la terminología necesaria. Los términos más importantes son *registro* y *campo* (Fig. 7.2). Un registro es un conjunto de datos referidos a un mismo elemento del fichero; es el concepto equivalente al de *ficha* en un fichero de tradicional. Por ejemplo, en un fichero de locomotoras de vapor, a cada modelo se le dedicaría un registro. Dentro de ese registro o ficha se reservaría espacio para cada una de las especificaciones de la máquina. Cada una de esas especificaciones es un campo. Por ejemplo, un campo sería el año de fabricación, otro el nombre del fabricante, etc. El conjunto de todos estos campos constituye un registro, y el conjunto de todos los registros forma el fichero.

SISTEMA DE ARCHIVO EN CINTA

En este libro, dado que disponemos del sistema de grabación del CPC464, podemos ignorar los métodos de archivo que se basan en las líneas DATA de un programa de BASIC. Esto es posible porque mediante los archivos en cinta se pueden almacenar los datos independientemente de los programas, lo cual, como hemos explicado,

es mucho más útil. Si todo esto le resulta conocido, tenga paciencia; ya llegaremos a algo que usted no haya visto aún.

Hay dos tipos generales de ficheros y sólo uno de ellos es posible en un sistema de grabación en cinta. Los dos tipos son los *ficheros secuenciales* y los *ficheros de acceso aleatorio*. La diferencia es sencilla, pero decisiva. En un fichero secuencial, la información se graba ordenadamente, con un dato tras otro. Si se quiere leer un dato, hay que leer el fichero desde el principio, hasta llegar al dato de interés. No hay una forma sencilla de programar la lectura de un dato aislado. En cambio, un

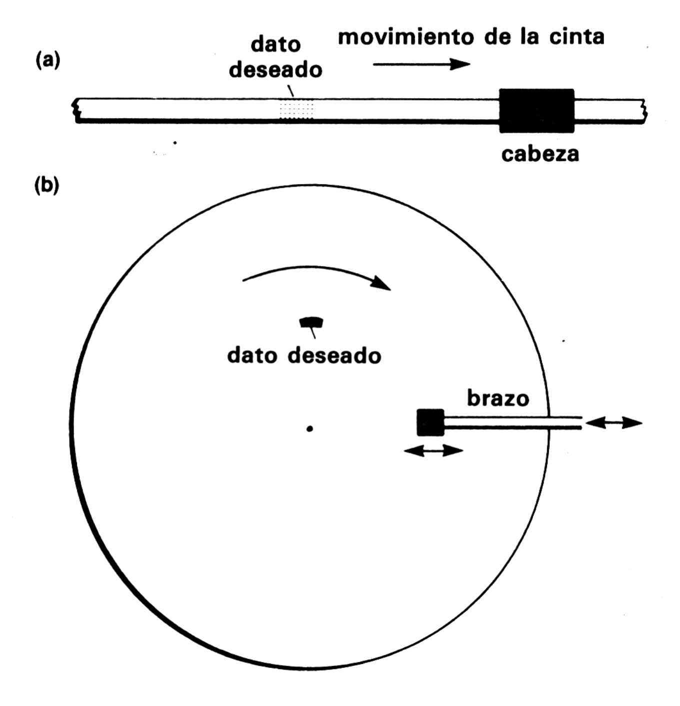


Fig. 7.3 Diferencia entre (*a*) ficheros secuenciales en cinta y (*b*) ficheros de acceso aleatorio en disco. Todos los puntos del disco son accesibles inmediatamente sin más que colocar la cabeza a la distancia del centro correcta. En cambio, para leer un dato de la cinta, hay que recorrer la cinta desde el principio del fichero hasta el lugar donde se encuentre el dato.

fichero de acceso aleatorio es lo que su nombre sugiere: un fichero del que se puede leer campo de cualquier registro sin tener que recorrer el fichero entero. En la figura 7.3 se ilustra la diferencia entre los dos métodos de archivo. Los sistemas de ficheros de acceso aleatorio requieren unidades de disco; no obstante, podemos diseñar programas que producen más o menos el mismo efecto utilizando ficheros secuenciales. Estudiemos, pues, este tipo de ficheros.

CREACIÓN DE UN FICHERO

Como siempre que abordamos algo nuevo, vamos a empezar por lo más sencillo. Veamos en primer lugar cómo establecer la conexión entre el ordenador y el magnetófono. El ejemplo de la figura 7.4 ilustra cómo se puede grabar un dato, concretamente el valor de la variable A; el número del canal que conecta el ordenador con el magnetófono es el 9. En la línea 10 *abrimos un fichero*. La instrucción es la palabra OPENOUT seguida del nombre del fichero entre paréntesis. El efecto de la línea 10 es, pues, abrir un fichero nuevo en la cinta con el nombre de «PRUEBA». Antes de empezar una grabación hay que estar seguro de que la posición en la cinta es la correcta. El sistema no puede impedir que al grabar se borre una grabación anterior. Los sistemas basados en discos protegen contra errores de este tipo, pero cuando se graba en cinta la única ayuda es el contador del magnetófono.

```
10 OPENOUT "PRUEBA"
20 A=5
30 PRINT#9,A
40 CLOSEOUT
```

Fig. 7.4 Grabación del valor de una sola variable. Lo que se graba es el valor, no el nombre de la variable.

El siguiente paso es asignar un valor a la variable A (línea 20). La línea 30 es la fundamental. La instrucción de esta línea significa «trasmitir por el canal número 9 el valor de A». Puesto que el canal 9 está permanentemente asignado al magnetófono, el efecto de la instrucción es «escribir» en la cinta. Durante la grabación aparecen en la pantalla los mismos mensajes que cuando se graban programas. El valor queda archivado con el nombre «PRUEBA», de forma que podrá ser encontrado fácilmente cuando se lo busque por ese nombre. La línea 30 hace la grabación, desde luego, pero su acción no es tan sencilla como en principio puede parecer. El sistema de grabación envía a la cinta los caracteres agrupados; el sistema operativo continúa agrupando caracteres hasta que se llena cierta zona de la memoria, denominada *tampón*. Cuando se abre un canal, el sistema operativo automáticamente reserva una zona de memoria para utilizarla como tampón. Al ejecutar una instrucción tal como la de la línea 30, los datos que han de ser grabados son transferidos al tampón. Si éste se llena (su capacidades de 2000 caracteres), los datos que contiene son enviados a la cinta; esto puede ocurrir varias veces en el curso de una misma grabación. Al final del proceso hay que asegurar que todos los datos hayan sido grabados, es decir, que el tampón esté vacío. Para ello sirve la instrucción CLOSEOUT de la línea 40.

Por complicadas que sean estas transferencias de datos, al usuario lo único que le interesa es que el ordenador escribe en la pantalla los mensajes habituales, que la cinta se mueve durante algún tiempo y después se para y, sobre todo, que los datos han quedado grabados correctamente. Para comprobar esto último hemos escri-

```
10 OPENIN "PRUEBA"
20 INPUT #9,X
30 PRINT"X ES":X
40 CLOSEIN
```

Fig. 7.5 Lectura de un dato grabado en cinta.

to el programa de la figura 7.5. En la línea 10 hemos puesto la instrucción OPENIN, no OPENOUT, porque lo que queremos ahora es leer el fichero que ya tenemos en la cinta, no crear uno nuevo. También tenemos que especificar el nombre del fichero, «PRUEBA». El número del canal sigue siendo, naturalmente, el 9. Una vez abierto un fichero, lo que significa que el sistema operativo reserva un tampón para almacenar provisionalmente los datos procedentes del magnetófono, empieza la búsqueda del fichero en la cinta. Si no se encuentra el fichero, el ordenador continúa buscando mientras quede cinta por leer. Los sistemas de discos realizan esta función de forma mucho más eficaz, porque pueden averiguar inmediatamente si existe en el disco un fichero con el nombre especificado. Una vez encontrado el fichero podemos leer los datos. De ello se encarga en este ejemplo la línea 20, con la instrucción INPUT #9,X. La palabra INPUT aislada hace referencia al teclado, pero seguida de #9 lee del magnetófono. Como hemos especificado el nombre «PRUEBA» en la instrucción OPENIN, de la cinta no pueden venir más datos que los que antes habíamos grabado con ese nombre. La línea 30 escribe en la pantalla el dato leído, y la línea 40 cierra el canal. Todo esto parece, y es, muy sencillo. Pero debemos observar estos dos programas muy atentamente porque continen la esencia de lo que vamos a necesitar en nuestro trabajo de archivo. Así, notemos que el dato leído puede ser asignado a una variable de nombre cualquiera. Hemos utilizado la variable A en la grabación, y la X en la lectura. Para el ordenador, leer de la cinta es sencillamente leer de un canal, y opera igual cualquiera que sea el número del canal.

La figura 7.6 es un ejemplo un poco más ambicioso, en el que generamos un fichero de números (los pares del 0 al 50) al tiempo que los grabamos en cinta y los escribimos en la pantalla. La línea 10 conecta un tampón con un canal. El canal es el número 9, el del magnetófono, y el nombre del fichero es «PARES». A continuación se crea un fichero, mediante el bucle que empieza en la línea 20. Este bucle asigna a la variable N los números pares del 0 al 50. ¿Cómo enviamos los números al tampón? De ello se encarga la línea 30, con la instrucción PRINT #9,N. Puesto que

```
10 OPENOUT "PARES"
20 CLS:FOR N=0 TO 50 STEP 2
30 PRINT #9,N
40 PRINT#0,N;" ";
60 CLOSEOUT
```

Fig. 7.6 Creación de un fichero de números y grabación de sus valores.

los datos no son suficientes para llenarlo, el tampón los acepta todos *antes* de que empiece la grabación. El sistema de grabación es muchísimo más lento que el ordenador. La ejecución del bucle FOR ... NEXT completo antes de que el motor del magnetófono haya llegado a ponerse en marcha. Cada vez que el bucle pasa por la línea 30, el valor actual de N es enviado al tampón. Si el número de datos fuera mayor, la grabación empezaría cuando se llenara el tampón; pero en este caso el bucle se completa antes de que eso ocurra. La grabación empieza cuando se ejecuta la instrucción CLOSEOUT, la cual hace, entre otras cosas, que el tampón se vacíe hacia el magnetófono y que detrás de los datos se grabe una señal de *fin de fichero*. Esta señal permite al ordenador identificar el final del fichero cuando más tarde lo lea. Si ejecuta este programa, verá cómo los números aparecen en la pantalla; el mensaje sobre pulsar las teclas REC y PLAY aparece *después* de completarse el bucle.

En algunas aplicaciones no es conveniente que aparezcan los mensajes habituales en la pantalla. Para suprimirlos basta con hacer que el primer carácter del fichero sea un signo de admiración (!). Por ejemplo, si en la lína 10 ponemos OPENOUT"!PARES", lo único que veremos en la pantalla serán los números. Esto es útil, porque permite al programador escribir en la pantalla otros mensajes.

Este programa ha demostrado el funcionamiento del tampón; su capacidad es suficiente para muchos más datos. Compruébelo con el programa de la figura 7.7. En él hemos puesto el signo de admiración en el nombre del fichero para suprimir los mensajes. De los mensajes nuevos se encargan las línenas 20 y 30. Esta vez los datos ocupan varios millares de bytes. Cuando ejecute el programa verá que en la pantalla aparecen los números hasta el 598. En ese momento el motor del magnetófono se pone en marcha y se realiza la grabación de un bloque. A continuación los números siguen apareciendo en la pantalla hasta llegar al 1160; entonces se graba el segundo bloque, y otro tanto sucede al llegar a 1672, y así sucesivamente cada 512 números. Lo que sucede es que el tampón esta siendo llenado por el bucle y vaciado hacia la cinta.

```
10 OPENOUT "!MASPARES"
20 PRINT"PULSE REC Y PLAY AHORA."
30 PRINT"PULSE LA BARRA ESPACIADORA PARA EMPEZAR."
40 WHILE INKEY(47)=-1:WEND
50 CLS:FOR N=0 TO 5000 STEP 2
60 PRINT #9,N
70 PRINT N
80 NEXT
90 CLOSEOUT
100 PRINT"FIN DE LA GRABACION. PULSE STOP."
```

Fig. 7.7 Un fichero de números mucho más largo con cuya grabación se puede observar el el funcionamiento del tampón. Los mensajes han sido suprimidos poniendo el signo ! al principio del nombre del fichero.

```
10 PRINT"PULSE LA TECLA PLAY Y LUEGO LA BARRA
   ESPACIADORA"
20 WHILE INKEY(47)=-1:WEND
30 OPENIN"!PARES"
40 FOR N=0 TO 50 STEP 2
50 INPUT#9, A
60 PRINT A;" ";
70 NEXT
80 CLOSE IN
90 PRINT:PRINT"PULSE LA TECLA STOP DEL
   MAGNETOFONO"
```

Fig. 7.8 Lectura del fichero PARES, con los mensajes habituales sustituidos por otros.

Estos pequeños programas han sido muy instructivos, pero al lector le puede quedar la duda de si efectivamente algo ha quedado grabado en cinta. Vamos a comprobarlo con el programa de la figura 7.8 para el caso del fichero «PARES». En este ejemplo hemos utilizado el nombre !PARES para suprimir los mensajes. Esto representa que tenemos que programar nuestros propios mensajes; pero, como demuestra el programa, hemos de cuidar dónde los ponemos. Si la primer instrucción fuera OPENIN«!PARES», el programa no parecería funcionar correctamente; esto es así porque al ejecutar esta instrucción la máquina tiene que buscar el fichero ''PARES'' (para la búsqueda es indiferente que haya sido grabado con o son signo de admiración) antes de poder hacer otra cosa. Por consiguiente, los nuevos mensajes han de ser escritos antes de que el programa llegue a OPENIN. El núcleo del programa empieza en la línea 30, en la que se abre mediante OPENIN un fichero con el nombre !PARES. De este fichero leemos datos con un bucle que además los escribe en la pantalla. Al ejecutar el programa, primero se oye que el motor se pone en marcha, luego el movimiento de la cinta mientras encuentra y lee el fichero, y finalmente los datos son visualizados en la pantalla.

Comprobemos ahora el fichero más largo, !MASPARES, con el programa de la figura 7.9. Hemos modificado la forma de visualizar los datos en la pantalla introduciendo un retardo cada vez que ésta se llena (línea 70). La condición es IF N/20 = INT(N/20), que se cumple cuando N es múltiplo de 20, pues éste es el único caso en el que la división no tiene resto y el cociente N/20 es igual a su parte entera. Así pues, cada vez que N es múltiplo de 20, la línea 70 produce un retardo, al final del cual se borra la pantalla y el programa continúa. Al ejecutar el programa, se oye cómo el motor se pone en marcha para buscar el fichero y leer en él. El número 0 aparece en la pantalla bajo el texto y a partir de ahí el bucle principal visualiza los números de 20 en 20. El motor del magnetófono se pone en marcha y se detiene cada vez que se vacíe el tampón. El proceso es bastante lento, aunque no tanto si los datos hubieran sido grabados a velocidad doble con la orden SPEED

```
10 PRINT"PULSE LA TECLA PLAY DEL MAGNETOFONO"
20 PRINT"Y LUEGO LA BARRA ESPACIADORA PARA EMPEZAR"
25 WHILE INKEY(47)=-1:WEND
30 OPENIN"!MASPARES"
40 FOR N=0 TO 5000
50 INPUT #9,K
60 PRINT K
70 IF N/20=INT(N/20) THEN FOR J=1 TO 2000:NEXT:CLS
80 NEXT
90 PRINT"PULSE STOP EN EL MAGNETOFONO"
```

Fig. 7.9 Lectura del fichero MASPARES. Se ha introducido un retardo para que dé tiempo a observar los números en la pantalla.

WRITE 1. El ordenador selecciona automáticamente la velocidad de lectura, que viene impuesta por la velocidad a la que se haya realizado la grabación. Obsérvese que en este programa falta la instrucción CLOSEIN. Esto no causa problema si se ejecuta el programa una sola vez y luego se inicializa la máquina con la orden NEW. Pero, de no ser así, no se podrá leer otro fichero con OPENIN mientras no se haya ejecutado una instrucción CLOSEIN.

OTROS FICHEROS SECUENCIALES

Supongamos que lo que se desea grabar no es un conjunto de números generado por un programa, sino un conjunto de nombres introducidos por el teclado. Para el sistema de grabación esto no representa nada nuevo, pues lo que graba es conjuntos de datos sin importale su naturaleza. Cada vez que se pulsa ENTER en una instrucción INPUT, los datos son enviados al tampón, y permanecen en él hasta que se llena o hasta que el proceso de captación termina y se cierra el canal. Nuevamente podemos apreciar aquí la necesidad de disponer de un tampón: no sería lógico que el motor del magnetófono tuviera que ponerse en marcha y detenerse cada vez que pulsáramos una tecla.

En la figura 7.10 se da un programa de este tipo. Normalmente, para recopilar información de este tipo habría que almacenar los nombres asignándolos a una lista, lo que obligaría a dimensionarla previamente. Pero esto no es necesario, a menos que se quiera poder consultar o modificar algún elemento introducido con anterioridad. Las cosas no serán tan fáciles cuando leamos el fichero, pero de eso ya hablaremos más tarde.

Analicemos el programa en detalle. La línea 10 selecciona la velocidad más alta con SPEED WRITE 1. Esto no significa que la cinta vaya a moverse más deprisa, si no que los datos son enviados al canal 9 a mayor velocidad. Cancelaremos este modo de grabación al final del programa, por si más tarde quisiéramos, por ejemplo, grabar un programa a la velocidad más lenta (y más fiable). La línea 20 abre

```
10 SPEED WRITE 1
20 OPENOUT "!NOMBRES"
30 CLS:PRINT TAB(18)"NOMBRES"
40 PRINT:PRINT"ESTE PROGRAMA GRABA UN FICHERO DE":
   PRINT"NOMBRES EN CINTA.":PRINT"PONGA UNA CINTA
   EN EL MAGNETOFONO."
50 PRINT"INTRODUZCA X PARA TERMINAR EL PROCESO."
60 PRINT"PULSE REC Y PLAY EN EL MAGNETOFONO, Y A":
   PRINT"CONTINUACION LA BARRA ESPACIADORA":PRINT
   "CUANDO ESTE PREPARADO."
70 WHILE INKEY(47)=-1:WEND
80 WHILE NOMBRE$<>"X" AND NOMBRE$<>"x"
90 INPUT"NOMBRE:";NOMBRE$
100 PRINT#9,NOMBRE$
110 WEND
120 CLOSEOUT
130 PRINT"PULSE STOP EN EL MAGNETOFONO."
140 SPEED WRITE 0
```

Fig. 7.10 Archivo de nombres. Se ha elegido la velocidad de grabación más alta para hacer el programa más eficaz.

el fichero, suprimiendo los mensajes como de costumbre. Las líneas 30 a 60 dan instrucciones al usuario. La línea 70 espera hasta que se pulse la barra espaciadora.

El bucle principal tipo WHILE empieza en la línea 80. Su función es captar y grabar en cinta los nombres que se introduzcan hasta que la cadena recibida sea "X" o "x". La línea 90 capta el nombre tecleado (recuerde que la instrucción INPUT no permite el uso de comas). La línea 100 lo graba. Si el nombre no era X ni x, la línea 110 reconduce el programa a la captación del siguiente nombre. Si los nombres se introducen deprisa y de forma continua, el magnetófono se pone en marcha de vez en cuando; mientras se está moviendo no se pueden introducir datos.

LECTURA DE FICHEROS

En este momento el lector tendrá grabados varios ficheros en una cinta, tanto de números como de nombres. Vamos a dedicar mayor atención a los métodos de lectura y a la posterior utilización de la información leída. Supongamos, por ejemplo, que los números que hemos grabado en el fichero PARES hubieran sido generados por un programa de contabilidad. Una cosa que podríamos necesitar sería leer los números del fichero para sumarlos. Ésta es una función habitual en informática. La figura 7.11 indica cómo realizarla. El programa empieza borrando la pantalla en la

```
10 CLS
20 PRINT TAB(18)"SUMA"
30 PRINT:PRINT"PULSE PLAY EN EL MAGNETOFONO":PRINT
   "CUANDO TENGA PREPARADA LA CINTA."
40 PRINT"PULSE LA BARRA ESPACIADORA PARA":PRINT
   "EMPEZAR LA LECTURA."
50 WHILE INKEY(47)=-1:WEND
60 OPENIN "!PARES"
70 TOTAL=0
80 FOR N=1 TO 26
90 INPUT #9,J
100 TOTAL=TOTAL+J
110 NEXT
120 PRINT:PRINT"LA SUMA ES";TOTAL
130 CLOSE IN
```

Fig. 7.11 Lectura del fichero PARES con un bucle FOR ... NEXT.

```
10 CLS
20 PRINT TAB(18)"SUMA"
30 PRINT:PRINT"PULSE PLAY EN EL MAGNETOFONO":PRINT
   "CUANDO TENGA PREPARADA LA CINTA."
40 PRINT"PULSE LA BARRA ESPACIADORA PARA":PRINT
   "EMPEZAR LA LECTURA."
50 WHILE INKEY(47)=-1:WEND
60 OPENIN "!PARES"
70 TOTAL=0
80 WHILE EOF=0
90 INPUT #9,J
100 TOTAL=TOTAL+J
110 WEND
120 PRINT:PRINT"LA SUMA ES";TOTAL
130 CLOSE IN
140 PRINT"PULSE LA TECLA STOP DEL MAGNETOFONO."
```

Fig. 7.12 Un método más general de lectura de un fichero. En este caso no es necesario saber de antemano cuántos datos contiene el fichero.

línea 10, escribiendo el título en la 20 y las instrucciones en las líneas 30 y 40. La línea 50 espera hasta que se pulse la barra espaciadora. La línea 60 abre el fichero !PARES. Recordemos que en este fichero habíamos grabado los números pares del 0 al 50, es decir, un total de 26 números. Para leerlos todos, el bucle FOR ... NEXT de la línea 80 tiene que ser ejecutado 26 veces. La suma de los datos ha sido puesta a cero en la línea 70. La línea 90 lee un número del fichero y lo asigna a la variable J. La línea 100 suma cada nuevo dato al total. Al final del programa, la línea 120 escribe en la pantalla la suma; la línea 130 cierra el fichero.

¿Qué ocurriría si no supiéramos cuántos números contiene el fichero? No podríamos utilizar un bucle FOR ... NEXT, pues no sabríamos que valores límite ponerle. La solución es muy fácil. El sistema de grabación del CPC464 escribe un indicador de fin de fichero al final del último bloque grabado. Ese indicador puede ser detectado mediante la palabra reservada EOF. Si EOF = 0, aún no se han acabado los datos; si EOF = 1, hemos llegado al final del fichero. Utilizando este recurso, el programa de lectura de PARES queda como se indica en la figura 7.12. Esta vez hemos puesto un buble WHILE ... WEND en lugar de un FOR ... NEXT. La condición para WHILE es EOF = 0, porque mientras sea cierto que EOF = 0 no habremos llegado al final del fichero.

LECTURA DE FICHEROS DE NOMBRES

Ya hemos aprendido grabar y leer un fichero de números; es hora de que aprendamos a los diversos métodos que podemos aplicar a la lectura del fichero de nombres que creamos antes. Recordemos que al crear ese fichero fuimos grabando los nombres uno tras otro y que al final del proceso el ordenador grabó el indicador de fin de fichero. La operación normal de lectura consistirá en leer nombres de la cinta y asignarlos a los elementos de una lista, de forma que el ordenador pueda manipularlos según convenga. La utilización de una lista nos permitirá llevar a cabo tareas tales como, por ejemplo, ordenar los nombres alfabéticamente. Pero las listas no siempre son necesarias. Supongamos, por ejemplo, que queremos buscar los nombres que empiecen por la letra J. En la figura 7.13 se da el programa que realiza este cometido.

Las primeras líneas no continen nada nuevo. La línea 60 capta la inicial por la que se deba buscar (no olvide escribirla en mayúsculas); la línea 70 abre el fichero; en la línea 80 empieza un bucle que lee un nombre del tampón, comprueba que no es el indicador de fin de fichero y compara la primera letra del nombre con la inicial especificada. Si no se encuentra el nombre buscado en el fichero, la línea 80 termina el programa cuando detecta el indicador de fin de fichero; si se encuentra, la línea 120 lo escribe en la pantalla. El bucle WHILE ... WEND es muy adecuado para la programación de esta búsqueda.

Este programa funcionará muy bien si los nombres no son muchos y con la condición de que sean distintos entre sí. Si en el fichero están los nombres «Manuel» y «Margarita», el programa escribe en la pantalla el primero que encuentre y da por terminada la búsqueda. Es necesario, pues, modificar las comprobaciones del bucle para que el programa escriba todos los nombres que empiecen con una letra dada.

```
10 CLS:PRINT TAB(16)"BUSCANOMBRES":PRINT
20 PRINT"ESTE PROGRAMA BUSCA NOMBRES":PRINT"POR LA
   INICIAL."
30 PRINT"PULSE PLAY CUANDO TENGA":PRINT"PREPARADA
   LA CINTA, Y LUEGO":PRINT"LA BARRA ESPACIADORA."
40 WHILE INKEY(47)=-1:WEND
50 SPEED WRITE 1
60 INPUT"ESCRIBA LA INICIAL DEL NOMBRE:";Q$
70 OPENIN "!NOMBRES"
80 WHILE EOF=0 AND Q$<>LEFT$(N$,1)
90 INPUT#9, N$
100 WEND
110 PRINT:PRINT"EL NOMBRE ES";N$
120 CLOSE IN:SPEED WRITE 0
130 PRINT"PULSE STOP EN EL MAGNETOFONO."
```

Fig. 7.13 Búsqueda de un dato en un fichero. En este caso, el nombre que empiece con una letra dada.

Muchos de los programas de este tipo funcionan menor si se almacenan todos los datos en memoria asignándolos a una lista. En tal caso el sistema de cinta no se utiliza más que como simple archivo, y todo el proceso de selección se realiza en la memoria del ordenador. Se preguntará entonces el lector si no sería más cómodo guardar los nombres en forma de líneas DATA en el programa. La ventaja de archivarlos por separado está en que de esa forma pueden ser utilizados por diferentes programas. Además, el programa que lee y manipula los datos puede ser bastante corto, de forma que deja espacio de memoria libre para muchos datos. Dicho de otra forma, el sistema de grabación del CPC464 permite manejar muchos datos con programas cortos.

El programa de la figura 7.14 lee los datos de un fichero y forma con ellos una lista. Cuando queremos construir una lista por este procedimiento nos topamos con un problema, que es cómo dimensionar la lista. Si los datos hubieran sido copiados de una lista a la cinta, la solución sería muy fácil: supongamos, por ejemplo, que

```
10 CLS:PRINT TAB(16)"BUSCANOMBRES"
20 PRINT:PRINT TAB(2)"CARGUE EL FICHERO DE NOMBRES
   SEGUN SE":PRINT"INDICA. CUANDO HAYA TERMINADO LA
   CARGA,"
30 PRINT"ESCRIBA LA INICIAL DE LOS NOMBRES QUE":
   PRINT"QUIERA VER (0 PARA TERMINAR)."
```

```
40 PRINT"REBOBINE LA CINTA HASTA EL PRINCIPIO DEL":
   PRINT"FICHERO NOMBRES. PONGA EL CONTADOR A"
50 PRINT"CERO. PULSE PLAY, Y LUEGO LA BARRA"
60 PRINT"ESPACIADORA CUANDO ESTE PREPARADO."
70 WHILE INKEY(47)=-1:WEND
80 PRINT:PRINT TAB(10)"POR FAVOR, ESPERE"
90 SPEED WRITE 1
100 OPENIN "!NOMBRES"
110 J=0:WHILE EOF=0
120 INPUT#9,N$
130 J=J+1
140 WEND:CLOSEIN
150 CLS:PRINT:PRINT"PULSE LA TECLA STOP"
160 PRINT:PRINT"AHORA REBOBINE LA CINTA HASTA 000.":
    PRINT"PULSE PLAY Y LA BARRA ESPACIADORA."
170 PRINT"CUANDO ESTE PREPARADO."
180 WHILE INKEY(47)=-1:WEND
190 PRINT:PRINT"POR FAVOR, ESPERE"
200 OPENIN "!NOMBRES"
210 DIM NOMBRE$(J)
220 FOR N=1 TO J
230 INPUT #9, NOMBRE$(N)
240 NEXT:CLOSE IN:SPEED WRITE 0
250 PRINT:PRINT"PULSE LA TECLA STOP"
260 FOR X=1 TO 2000:NEXT
270 CLS:PRINT:PRINT"ESCRIBA LA INICIAL DEL NOMBRE:
    ":Q$="X":M=0
280 WHILE Q$<>"0"
290 INPUT Q$:Q$=LEFT$(Q$,1):M=0
300 FOR X=1 TO J
310 IF Q$=LEFT$(NOMBRE$(X),1) THEN PRINT NOMBRE$(X):
    M=1
320 NEXT
330 IF M=0 AND Q$<>"0" THEN "NO ENCUENTRO EL
    NOMBRE.":PRINT"PRUEBE CON OTRO."
340 WEND
350 PRINT"FIN"
```

Fig. 7.14 Lectura de un fichero para construir una lista. El fichero se lee una vez para saber cuántos elementos contiene y poder dimensionar la lista, y por segunda vez para formar la lista.

las instrucciones INPUT asignan los datos a la lista Nombre$(J), en lugar de grabarlos directamente. Inmediatamente se nos ocurre que lo primero que tenemos que grabar al abrir el fichero es el valor de J, con PRINT # 9,J, y a continuación los datos. Ahora bien, si la grabación no se hizo a partir de una lista y no tuvimos la precaución de contar los datos, ¿qué podemos hacer? Una posibilidad es confiar en que el usuario sepa cuántos datos quiere leer, e introducir el número como respuesta a una pregunta del programa:

INPUT"Cuantos nombres?";N:DIM Nombre$(N)

Otra posibilidad es utilizar un contador en el programa que capta los datos:

INPUT Nombre$:N = N + 1

y grabar este número en un fichero distinto a continuación del fichero NOMBRES. Por ejemplo, un fichero podría ser NOMBREAMIGOS, y el otro NUMEROAMIGOS. Esto parece latoso, pero algún precio hay que pagar si se quiere dimensionar la lista con precisión. La precisión al dimensionar la lista sirve para evitar un error si tratáramos de poner demasiados elementos en ella y para no desperdiciar memoria sobredimensionándola.

Pero supongamos que tenemos que utilizar el fichero NOMBRES tal como está, es decir, sin saber cuántos nombres contiene. Tendremos que tomar medidas drásticas. Éste es otro aspecto en el que los discos aventajan notablemente a las cintas: en un disco se puede grabar un número después de los datos, y el número se puede leer cuando haga falta, antes o después que los datos. La figura 7.14 muestra un programa que resuelve este problema y no es demasiado lento. La idea conste en leer todos los datos, contándolos al mismo tiempo, hasta que se encuentra el indicador de fin de fichero. A continuación se pide al usuario que rebobine la cinta. Conocido el número de datos, ya de puede dimensionar la lista correctamente. Se vuelve a leer el fichero, esta vez asignando los datos a la lista. La selección de los nombres es entonces muy sencilla, pues no hay que volver a leer de la cinta.

Las líneas 10 a 60 escriben las instrucciones; la línea 70 es el detector de la barra espaciadora. En la línea 80 se escribe un mensaje para informar al usuario que se están cargando los datos. Se pone a cero la variable J. En cada pasada, el bucle lee un dato e incrementa el valor del contador. En la línea 140 cerramos el fichero, para poder abrirlo más tarde. Ahora necesitamos que el usuario rebobine la cinta, para lo cual le damos las instrucciones adecuadas. En la línea 210 utilizamos el valor de J para dimensionar la lista, antes de empezar a leer datos. Como ahora ya sabemos cuántos datos vamor a leer, podemos utilizar un bucle FOR ... NEXT (líneas 220 a 240) para leer y asignar nombres a la lista. Una vez completo este proceso, entra en acción el buscador de nombres (línea 280 a 350). Nótese que necesitamos dar un valor a la variable Q$ antes de entrar en el bucle. Cuando se encuentra un nombre se asigna el valor 1 a la variable M. Si no encuentra ningún nombre que empiece con la letra especificada, M sigue siendo 0 y se emite el mensaje de la línea 330. Cuando el usuario introduce un 0 para terminar la función de búsqueda, M también es cero, pero el mensaje no se escribe porque no se cumple la segunda parte de la condición, Q$< >"0". Es decir, si Q$="0" se suprime el mensaje.

MODIFICACIONES

Dejemos claro desde el principio que no es posible alterar el contenido de un fichero grabado en cinta. Lo que sí se puede hacer es leer el fichero, modificarlo y volverlo a grabar. La segunda versión del fichero se puede grabar con el mismo nombre en otra cinta o en otro lugar de la misma cinta. Si se conserva el mismo nombre, el fichero puede ser vuelto a leer por los mismos programas; esto no es tan fácil en los sistemas de disco. La técnica que se sigue en la figura 7.15 es muy similar a la de la figura 7.14.

Las líneas 10 a 260 son casi iguales a las de la figura 7.14; sólo se distinguen en el texto de las instrucciones. Su tarea principal es dimensionar la lista y formarlas con los datos leídos. Sin embargo, en este caso la línea 140 contiene $J = J - 1$. De esta forma se evita la lectura del último elemento de la lista, que es un finalizador: "x", "0" o el carácter que se utilizara cuando se creó el fichero.

Las líneas 270 preparar la grabación de la versión actualizada. A menos que el fichero sea largo, el usuario no nota la acción de las líneas 320 a 340, porque lo único que se estará produciendo es la transferencia del fichero al tampón, pero no la grabación. A continuación se pide al usuario que introduzca otros nombres. Cuando los esté introduciendo puede ocurrir que el tampón se llene; entonces se pondrá en marcha el magnetófono y mientras se esté moviendo no se podrá seguir escribiendo. En cambio, si se introducen pocos nombres, el tampón no se llenará, y por lo tanto no se oirá el funcionamiento del motor mientras no se escriba el 0 y termine el proceso. Así queda grabada la nueva versión del fichero. Para comprobarlo, ejecute otra vez el programa. Cuando el ordenador le pida que escriba más nombres, pulse ESC dos veces, teclee la línea

FOR Z = 1 TO J:?Nombre$(Z);" ";:NEXT

y pulse ENTER. En la pantalla aparecerá la lista de todos los nombres. Si quiere ver la lista organizada de otra forma, por ejemplo, en orden alfabético, tendrá que ampliar el programa con una rutina que realice esta nueva función.

```
10 CLS:PRINT TAB(10)"ACTUALIZACION DE FICHEROS"
20 PRINT:PRINT TAB(2)"CARGUE EL FICHERO DE NOMBRES
   SEGUN SE":PRINT"INDICA. CUANDO TERMINE LA CARGA
   SIGA"
30 PRINT"LAS INSTRUCCIONES PARA REGRABARLO.":PRINT
   "AHORA PUEDE AMPLIAR EL FICHERO CON OTROS"
40 PRINT"NOMBRES. ESCRIBA 0 PARA TERMINAR.":PRINT
   "REBOBINE LA CINTA HASTA EL PRINCIPIO DEL"
50 PRINT"FICHERO. PONGA EL CONTADOR A CERO.":PRINT
   "PULSE PLAY, Y LUEGO LA BARRA ESPACIADORA"
60 PRINT"CUANDO ESTE PREPARADO."
70 WHILE INKEY(47)=-1:WEND
80 PRINT:PRINT TAB(10)"POR FAVOR, ESPERE"
90 SPEED WRITE 1
100 OPENIN "!NOMBRES"
```

```
110 J=0:WHILE EOF=0
120 INPUT#9,N$
130 J=J+1
140 WEND:CLOSEIN:J=J-1
150 CLS:PRINT:PRINT"PULSE LA TECLA STOP"
160 PRINT"AHORA REBOBINE LA CINTA HASTA 000.":PRINT
    "PULSE PLAY Y LUEGO LA BARRA ESPACIADORA"
170 PRINT"CUANDO ESTE PREPARADO."
180 WHILE INKEY(47)=-1:WEND
190 PRINT:PRINT TAB(10)"POR FAVOR, ESPERE"
200 OPENIN "!NOMBRES"
210 DIM NOMBRE$(J)
220 FOR N=1 TO J
230 INPUT #9,NOMBRE$(N)
240 NEXT:CLOSE IN
250 PRINT:PRINT"PULSE LA TECLA STOP"
260 FOR X=1 TO 2000:NEXT
270 CLS:PRINT:PRINT TAB(5)"PREPARESE PARA VOLVER A
    GRABAR":PRINT"ESTE FICHERO"
280 PRINT:PRINT"BUSQUE UN SITIO LIBRE EN LA CINTA O":
    PRINT"PREPARE UNA NUEVA. PONGA A CERO EL"
290 PRINT"CONTADOR Y SIGA LAS INSTRUCCIONES.":PRINT
    "PULSE REC Y PLAY, Y LUEGO LA BARRA"
300 PRINT"ESPACIADORA CUANDO ESTE PREPARADO."
310 WHILE INKEY(47)=-1:WEND
320 OPENOUT "!NOMBRES"
330 FOR N=1 TO J
340 PRINT #9,NOMBRE$(N):NEXT
350 CLS:PRINT:PRINT TAB(16)"NOMBRES NUEVOS"
360 PRINT"INTRODUZCA LOS NOMBRES QUE DESEE.":PRINT
    "(0 PARA TERMINAR)."
370 NOMBRE$="X":WHILE NOMBRE$<>"0"
380 INPUT NOMBRE$
390 PRINT #9, NOMBRE$
400 WEND
410 SPEED WRITE 0
420 CLOSEOUT
430 PRINT"FIN"
```

Fig. 7.15 Actualización de un fichero. Puesto que la cinta no se puede modificar, hay que leer el fichero completo, actualizarlo y luego volverlo a grabar.

8

Ventanas y otros efectos

Trabajar con un ordenador nunca es aburrido, pero hay recursos con los que se puede hacer el trabajo mucho más interesante. Uno de los efectos especiales que ofrecen los ordenadores modernos es la posibilidad de utilizar ventanas. En este capítulo vamos a examinar ese efecto y algunos otros que permiten organizar la pantalla de forma mucho más atractiva que la mera visualización de texto y columnas de números. Las ventanas son un buen punto de partida, pues es muy probable que el lector no haya experimentado con ellas, a menos que antes haya tenido un ordenador capaz de manejarlas.

Una *ventana* no es más que una región de la pantalla que puede ser utilizada independientemente de las demás y de la pantalla en su conjunto. En una ventana se puede escribir texto, se la puede desplazar por *scrolling* o se la puede borrar, con independencia de lo que esté ocurriendo en el resto de la pantalla. Vamos a comprobarlo. Llene la pantalla con el listado de un programa o con algún otro texto y luego teclee la siguiente orden:

WINDOW 15,25,2,6 (ENTER)

Teclee ahora CLS. Sólo una pequeña región de la pantalla, que es la ventana que acabamos de definir, se ha borrado. Escriba un pequeño programa en esta ventana; observará que se comporta como si fuera la pantalla entera en cuanto al salto a la línea siguiente y en cuanto al desplazamiento del texto cuando se llega a la línea inferior. Mientras tanto, el resto de la pantalla sigue inalterado. Una orden CLS borrará solamente la ventana, no el resto de la pantalla. Para volver rápidamente a la situación normal teclee MODE 1 y pulse ENTER.

Si esto le ha despertado el apetito, pasemos a estudiar cómo podemos controlar el funcionamiento de las ventanas. La palabra WINDOW debe ir seguida de cuatro números que especifican, en este orden, la posición de los bordes izquierdo, derecho, superior e inferior de la ventana. Una vez ejecutada esta orden, todo el texto que el ordenador envíe a la pantalla irá a parar a esta ventana. Su número de canal es el 0, es decir, el canal normal para instrucciones PRINT y similares. Ahora bien, en la instrucción WINDOW se puede incluir un número de canal, por ejemplo, #2. Es tal caso la ventana sólo se utiliza cuando se ejecuta una instrucción tal como PRINT #2.

La figura 8.1 da un ejemplo del funcionamiento de una ventana. Primero se borra la pantalla entera y luego se abre una ventana con número de canal 2 mediante la instrucción

WINDOW #2,10,30,1,5

De esta forma, el canal número 2 queda conectado con la ventana especificada. Los cuatro últimos números de la instrucción definen tanto la posición como el tamaño de la ventana: su borde izquierdo estará en la columna 10, su borde derecho en la 30, su borde superior en la fila 1 y su borde inferior en la fila 5. Una vez ejecutada esta instrucción, se puede borrar la ventana (CLS #2) o escribir en ella (PRINT #2). Mientras no cerremos este canal o lo asignemos a algún otro

```
10 CLS
20 WINDOW#2,10,30,1,5
30 WINDOW#3,12,28,10,15
40 PRINT#2,"ESTA ES LA VENTANA #2"
50 GOSUB 150
60 PRINT#3,"OBSERVE EL TEXTO DE LA VENTANA #2. ESTA
   ES LA VENTANA #3"
70 GOSUB 150
80 CLS#2
90 GOSUB 150
100 CLS#3
110 GOSUB 150
120 PRINT#2,"VENTANA #2"
130 PRINT#3,"VENTANA #3"
140 END
150 FOR N=1 TO 2000:NEXT:RETURN
```

Fig. 8.1 Definición de ventanas en los canales número 2 y 3.

dispositivo, la ventana seguirá siendo controlada como canal #2. A pesar de todo, la pantalla entera puede seguir siendo utilizada. Además, si se borra la pantalla entera se borra también el contenido de la ventana; si se escribe en un lugar de la pantalla que coincida con la posición de la ventana, los caracteres nuevos sustituyen al anterior contenido de la ventana. Es decir, la pantalla entera *se solapa* con todas las ventanas, y éstas pueden solaparse entre sí, dependiendo de cómo estén definidas.

En este ejemplo las ventanas se definen en las líneas 20 y 30. Para demostrar su existencia, la línea 40 escribe una frase en la ventana #2. (Otra comprobación podría consistir en incluir en el programa la instrucción LIST #2, pero la máquina no ejecuta ninguna instrucción posterior a LIST.) En la línea 50 hay una pausa (el retar-

do de la subrutina de la línea 150). A continuación se escribe una frase en la ventana #3. El resto del programa ilustra cómo se puede escribir en las ventanas y borrarlas independientemente unas de otras. Nótese que al terminar el programa el mensaje «Ready» y el cursor quedan en el extremo superior izquierdo de la pantalla, es decir, en el lugar que les corresponde en la venta #0.

Otro efecto llamativo se consigue con la instrucción WINDOW SWAP. Supongamos, por ejemplo, que estamos utilizando un programa que capta datos y los graba en cinta, del tipo de los estudiados en el capítulo anterior. Podríamos utilizar una ventana para la entrada de datos y otra para la emisión de mensajes: por ejemplo, para enviar a ella los mensajes habituales del sistema de grabación. Si definimos una ventana con el número 3 en el extremo inferior de la pantalla y ejecutamos la instrucción

WINDOW SWAP #0, #3

todo lo que normalmente se escribiría en el canal #0 será enviado ahora al canal #3. La figura 8.2 simula esta función. No hemos desarrollado completamente las rutinas que afectan a la grabación para no complicar las cosas demasiado. Por otra parte, si no se quiere borrar el contenido de una cinta, lo que se puede hacer es pulsar PLAY en lugar de REC y PLAY cuando la máquina así lo pida. Otra forma de «engañar» al sistema de grabación es la siguiente: normalmente no se puede pulsar la tecla REC cuando no hay una cinta instalada en el magnetófono; en efecto, en el lugar en el que se aloja la cinta hay una palanca que detecta la presencia de ésta. Para poder pulsar REC basta con empujar esta palanca suavemente. De esta manera se pueden simular grabaciones sin necesidad de consumir ni estropear cintas.

Volviendo al programa, en las líneas 20 y 30 se definen dos ventanas. La altura de la ventana #3 es de sólo una línea y está situada cerca al fondo de la pantalla. De esta forma los mensajes se visualizan uno a la vez, con lo que evitan confusiones

```
10 CLS
20 WINDOW#2,2,39,5,10
30 WINDOW#3,2,39,25,25
40 PRINT#3,"INTRODUZCA LOS NOMBRES (X PARA TERMINAR)"
50 WHILE NOMBRE$<>"X" AND NOMBRE$<>"x"
60 INPUT NOMBRE$
70 WEND
80 WINDOW SWAP 0,3
90 OPENOUT "NOMBRES"
100 PRINT#9,"NOMBRE$"
110 CLOSEOUT
120 REM:PULSE ESC PARA DETENER EL PROGRAMA
```

Fig. 8.2 Utilización de WINDOW SWAP para confinar los mensajes a un lugar determinado de la pantalla.

con mensajes anteriores. Por supuesto, hay que estar seguros de que los mensajes no necesitan más de una línea. La línea 80 intercambia la ventana #0 con la #3. A partir de ese momento todos los mensajes aparecen en la ventana #3. Esta es una forma sencilla de mantener los mensajes separados del texto.

COLORES

Ha llegado el momento de empezar a manejar los colores del CPC464. Hay cuatro instrucciones fundamentales a este respecto: BORDER (borde), PAPER (papel), PEN (pluma), INK (tinta). Empezaremos por la más sencilla: BORDER. Esta ins-

Número	Color
0	Negro
1	Azul
2	Azul brillante
3	Rojo
4	Magenta (rojo + azul)
5	Malva
6	Rojo brillante
7	Morado
8	Magenta brillante
9	Verde
10	Cyan
11	Azul celeste
12	Amarillo
13	Blanco
14	Azul pastel
15	Naranja
16	Rosa
17	Magenta pastel
18	Verde brillante
19	Verde marino
20	Cyan brillante
21	Verde lima
22	Verde pastel
23	Cyan pastel
24	Amarillo brillante (dorado)
25	Amarillo pastel
26	Blanco brillante

Fig. 8.3 Los números de los colores. En un televisor o monitor monocromático los colores aparecen como gradación de tonos grises, tanto más brillantes cuanto más alto es su número.

trucción puede ir seguida de uno o dos números. Si sólo se pone un parámetro, se habrá especificado un color constante para el borde de la pantalla. (Los números de los colores son los se muestran en la tabla de la figura 8.3) Si se ponen dos parámetros, el color del borde alternará entre los dos colores especificados.

```
10 REM LISTE EL PROGRAMA ANTES DE EJECUTARLO
20 WINDOW#2,35,40,5,5
30 FOR N=0 TO 26
40 PRINT#2,N
50 BORDER N
60 GOSUB 150
70 NEXT
80 GOSUB 150
90 FOR N=1 TO 26
100 PRINT#2,N;",";27-N
110 BORDER N,27-N
120 GOSUB 150
130 NEXT
140 END
150 EMP=TIME
160 WHILE TIME<EMP+600:WEND
170 RETURN
```

Fig. 8.4 Visualización de los colores en el borde de la pantalla. El retardo se controla con la función TIME.

El programa de la figura 8.4 es un primer ejemplo de utilización de los colores. Lo primero que hace es definir una ventana, en el extremo superior derecho de la pantalla, en la que después se van a mostrar los colores. A continuación se entra en un bucle que va cambiando el color del borde. Con cada cambio de color se introduce un retardo. A diferencia de lo que hacíamos en ejemplos anteriores, aquí controlamos el retardo mediante el cronómetro interno de la máquina. TIME es una magnitud que parte del valor cero cuando se enciende la máquina y se incrementa en 300 unidades cada segundo. El funcionamiento de la subrutina 150-170 es el siguiente. A la variable EMP se le asigna el valor que tiene TIME en el momento de ejecutar la instrucción. Al cabo de un segundo, el valor de TIME habrá aumentado en 300; el bucle WHILE ... WEND espera hasta que el valor de TIME exceda del de EMP en 600, es decir, hasta que hayan transcurrido 2 segundos. Es un método sencillo y elegante, y más preciso para la medida del tiempo que los bucles FOR ... NEXT. Volvamos al programa. Una vez exhibidos los colores aislados, se empieza a visualizarlos alternadamente. La frecuencia de parpadeo se controla mediante otra variable que se puede variar a voluntad. Al final del programa, mientras los colores

aún están parpadeando, teclee la orden SPEED INK 10,10. Al pulsar ENTER observará que el parpadeo es más rápido. Teclee a continuación SPEED INK 20,100 y observe el efecto. Los números especifican el tiempo que debe ser visualizado cada uno de los dos colores; cada unidad es un cincuentavo de segundo (es decir, el número 50 representa un segundo). Tenga precaución con esta instrucción: *(las velocidades próximas a 8 parpadeos por segundo pueden ser peligrosas para los epilépticos.*

Lejos de ser agradable, el parpadeo del borde puede resultar muy molesto. Uno de los efectos del parpadeo es alterar ligeramente el tamaño de la pantalla útil. La razón es que ni los televisores ni los monitores han sido diseñados para manejar correctamente imágenes parpadeantes (un monitor que pudiera hacerlo costaría demasiado caro, quizá más de 200 000 pesetas). Para detener el parpadeo del borde teclee la orden BORDER 1. Un cambio de modo de pantalla (MODE 1) no afectará al borde.

Las siguientes instrucciones son PAPER, PEN e INK. Los nombres de estas instrucciones dan idea de su funcionamiento, pero también pueden crear confusión, en particular si el lector ha hecho su primer aprendizaje en un Spectrum. Vamos a explicarlas, empezando por INK.

INK representa un color de tinta. El número de colores de que se dispone depende del modo de pantalla. En modo 0 se pueden utilizar simultáneamente hasta 16 colores de tinta, elegidos de entre los 27 posibles que se relacionan en la figura 8.3. En modo 1 los colores de tinta disponibles son cuatro; también en este caso podemos elegir, de los 27, qué cuatro colores vamos a utilizar. En modo 2 los colores de tinta sólo pueden ser dos (a elegir de entre los 27). Para comprender el significado de INK imaginemos que disponemos de una pluma y varios tinteros (16, 4 o 2). En los tinteros podemos poner la tinta que queramos, eligiendo de un muestrario de 27 colores diferentes: pero a la hora de escribir sólo disponemos de las tintas que haya en los tinteros; para cambiar algún color tenemos que vaciar el tintero y rellenarlo con otra tinta.

A partir de aquí el CPC464 empieza a ser diferente de otras máquinas. Así, en modo 1 los *números* de tinta (INK) son 0, 1, 2 y 3. Observe que hemos dicho *números*, no *colores*. Volviendo al símil de los tinteros, estos números serían los que están marcados en el vidrio de los tinteros. La instrucción INK nos permite llenar los tinteros con la tinta deseada. Por ejemplo, INK 2,7 llena el tintero número 2 con tinta de color 7 (morado). Cuando se enciende el ordenador, o cada vez que se cambia de modo, la máquina llena los tinteros con los colores *implícitos* o *iniciales*. En el tintero número 0 pone tinta 1 (azul); ésta es la tinta con la que pinta el fondo de de la pantalla. En el tintero 1 pone tinta 24 (dorada), que es la tinta con la que escribe los caracteres. En la figura 8.5 damos la relación de los colores iniciales.

Modos 0 y 1

Tintero 0	...	color 1	...	azul
Tintero 1	...	color 24	...	dorado
Tintero 2	...	color 20	...	cyan brillante
Tintero 3	...	color 6	...	rojo brillante

(En modo 2 sólo se pueden utilizar dos colores, inicialmente el 1 y el 24.)

Fig. 8.5 Colores de tinta implícitos.

Podemos cambiar estos colores iniciales con la instrucción INK. En modo 1 disponemos de los tinteros 0 al 3 (podemos utilizar números mayores, pero la máquina los convierte internamente al margen 0-3). Para asignar la tinta de color 15 al tintero número 1, haremos INK 1,15. Si especificamos dos colores, el tintero contendrá una extraña tinta que parpadea pasando alternadamente del uno al otro. Por ejemplo, INK 1,3,7 pone en el tintero número 1 una tinta que parpadea entre los colores 3 y 7. Se puede modificar la frecuencia de parpadeo mediante la instrución SPEED INK.

Al encender el ordenador, automáticamente se selecciona el modo 1, y los cuatro tinteros se llenan con las tintas indicadas en la figura 8.5. La máquina elige también en ese momento qué tinteros selecciona para pintar el fondo de la pantalla y para escribir en ella. Para el fondo (PAPER) toma el tintero número 0, es decir, PAPER 0. Para escribir los caracteres toma el tintero número 1, o sea, PEN 1. Recordemos una vez más que estos números no tienen nada que ver con los colores, sino con los tinteros. Con la instrucción PAPER podemos especificar de qué tintero queremos que se tome la tinta para pintar el fondo de la pantalla. Análogamente, PEN permite elegir el tintero del que se va a tomar la tinta para escribir los caracteres.

En la figura 8.6 se ilustra el manejo de estas instrucciones. Si tecleamos una orden tal como PAPER sin más, no observaremos ninguna alteración en la pantalla. Sólo

```
10 BORDER 0
20 FOR N=0 TO 3
30 PAPER N:CLS
40 GOSUB 140
50 NEXT
55 PAPER 0:CLS
60 FOR N=0 TO 3
70 PEN N
80 GOSUB 170
90 GOSUB 140
100 NEXT
110 GOSUB 140
120 PEN 1
130 END
140 EMP=TIME
150 WHILE TIME<EMP+600:WEND
160 RETURN
170 PRINT"ESTE TEXTO ILUSTRA EL EFECTO DEL COLOR DE
    LA PLUMA (PEN):";N
180 RETURN
```

Fig. 8.6 Utilización de las instrucciones PAPER y PEN.

notaremos el efecto cuando continuemos escribiendo, pues entonces los nuevos caracteres aparecerán sobre fondo de color cyan brillante. Para que PAPER 2 afecte a la pantalla entera tiene que ir seguida de CLS. Esto se demuestra en la línea 30. En la segunda parte del programa utilizamos PAPER 0, que es azul oscuro (por ser éste el color que tenemos en el tintero número 0). Por los resultados de este programa se puede observa que para optimizar la claridad del texto es necesario que los colores del papel (fondo) y de la pluma estén bien contrastados. Mi preferencia es un color oscuro para el papel y un color claro para la tinta, pues lo opuesto, texto oscuro sobre fondo claro produce un efecto de inestabilidad que me resulta muy irritante. No es de esperar que los colores de las letras sean muy buenos en un televisor, pues los televisores de color no están diseñados para esta tarea específica. Si además tenemos en cuenta que el 90 por 100 de los varones no discrimina perfectamente los colores, se comprende que los mejores efectos en color se consigan solamente en las condiciones más favorables: con un monitor y utilizando colores fuertes en superficies grandes.

Al ejecutar este programa se observa que con PEN 0 no aparece el texto. La razón es que la tinta del tintero número 0 es la misma que se ha utilizado para el fondo (línea 55). En este ejemplo hemos elegido el modo 1. Recordemos, no obstante, que en modo 2 se dispone de dos tinteros, el 0 y el 1, y en modo 0 de 16 tinteros, del 0 al 15.

Las instrucciones PEN y PAPER pueden ser aplicadas, no sólo a la pantalla entera, sino también a las ventanas. La figura 8.7 ilustra este hecho, y también que no todas las combinaciones de colores son buenas. Antes de asignar colores de papel

```
10 BORDER 0
20 WINDOW#1,1,40,3,5
30 WINDOW#2,1,40,7,10
40 WINDOW#3,1,19,11,24
50 WINDOW#4,20,40,11,24
60 A$="PRUEBA PARA MOSTRAR LAS VENTANAS"
70 PAPER#1,0
80 PAPER#2,1
90 PAPER#3,2
100 PAPER#4,3
110 PEN#1,3
120 PEN#2,2
130 PEN#3,1
140 PEN#4,0
150 CLS
160 FOR N=1 TO 4
170 PRINT#N,A$:NEXT
```

Fig. 8.7 PAPER y PEN en diferentes ventanas.

y pluma a las ventanas, compruebe que los colores son compatibles. A veces las combinaciones de dos tonos de un mismo color son atractivas, pero en general no es aconsejable combinar dos colores claros o dos colores oscuros, especialmente cuando se trabaja con un televisor. Algunas combinaciones no son adecuadas ni siquiera para un monitor.

ESCRITURA MÁS ATRACTIVA

Vamos a explorar las posibilidades de hacer más atractiva la visualización de textos, y para ello comenzaremos otra vez con la instrucción INK. El programa de la figura 8.8 hace parpadear un texto. El parpadeo se consigue poniendo mezclas de dos tintas en los tinteros 2 y 3 (líneas 20 y 30). Cuando mojamos la pluma en estos tinteros, el texto que escribimos parpadea (líneas 50 y 60). La línea 90 retorna a PEN 1, con objeto de que el mensaje «Ready» no parpadee. Obsérve que las letras siguen parpadeando mientras estén en la pantalla. En todo momento se pueden cambiar los colo-

```
10 CLS
20 INK 2,6,8
30 INK 3,12,19
40 PEN 2
50 PRINT:PRINT"ATENCION, POR FAVOR!"
60 PRINT:PRINT:PRINT
70 PEN 3
80 PRINT"PRECAUCION: PELIGRO!"
90 PEN 1
```

Fig. 8.8 Texto parpadeante que se consigue mezclando dos tintas en un tintero.

res (o poner un solo color para que no parpadee) alterando la tinta mediante INK. Por ejemplo, la figura 8.9 ilustra cómo se puede hacer que un título parpadee durante un solo segundo y luego se quede fijo en otro color. De esta manera se consigue que el parpadeo atraiga la atención al usuario sin molestarlo.

El CPC464 permite la realización de efectos especiales, tales como subrayar letras y mezclarlas con acentos. Para ello se utiliza uno de los códigos «no imprimibles» del CPC464. Estos códigos son los comprendidos entre el 1 y el 31. Ya hemos utilizado antes el 21 (para apagar la pantalla), el 6 (para encerderla) y el 8 (para mover el cursor hacia la izquierda). El manual da una lista completa de las funciones de estos códigos en el capítulo 9, página 2. Para subrayar texto, como se hace en el programa de la figura 8.10, lo primero que necesitamos es cambiar el modo de escritura de los caracteres. Normalmente, al escribir un carácter, el nuevo reemplaza completamente al anterior. Con CHR$(22) + CHR$(1) ponemos la pantalla en mo-

```
10 INK 2,24,6
20 CLS
30 PRINT:PEN 2
40 PRINT TAB(14)"TITULO PULSATIL"
50 FOR N=1 TO 3000:NEXT
60 INK 2,20
70 PEN 1
80 PRINT:PRINT"AHORA PODEMOS ESCRIBIR EN LA PANTALLA"
   :PRINT"SIN QUE EL PARPADEO NOS DISTRAIGA"
```

Fig. 8.9 Supresión del parpadeo cambiando la tinta con INK.

```
10 CLS
20 PRINT:PRINT:PRINT
30 PRINT"ESTO ES IMPORTANTE"
40 S$=STRING$(18,8)
50 PRINT S$;
60 PRINT CHR$(22)+CHR$(1);
70 PRINT"------------------"
80 PRINT:PRINT
90 PRINT CHR$(22)+CHR$(0)
```

Fig. 8.10 Subrayado de textos. CHR$(22) permite la superposición de caracteres. Obsérvese la utilización de STRING$ para hacer retroceder el cursor 18 veces.

do «transparente», lo que permite superponer un texto a otro. En el programa se escribe la frase de la línea 30. Escribiendo la cadena S$ hacemos volver al cursor a su posición original; la línea 60 cambia el modo de escritura, y la línea 70 escribe el subrayado. El modo transparente es muy molesto y confuso cuando estamos escribiendo o modificando un programa; con CHR$(22) + CHR$(0) volvemos al modo opaco. Muchas de estas órdenes necesitan dos o más términos CHR$; la forma que hemos indicado es la que permite combinarlos para formar una orden completa. Algunas de estas órdenes son equivalentes a órdenas de BASIC. La ventaja de disponer de ellas en esta forma es que permite realizar las funciones mediante el uso de números.

OTRAS INSTRUCCIONES

Hay otras muchas órdenes (o instrucciones) que no son fáciles de clasificar. Tal es el caso de LOWER$ y UPPER$ (Fig. 8.11). Ambas deben ir seguidas de una cadena literal entre paréntesis. El efecto es el siguiente: UPPER$ convierte las letras de la

```
10 CLS:PRINT:PRINT TAB(10"MAYUSCULAS-MINUSCULAS"
20 PRINT:PRINT
30 FOR N=1 TO 5
40 INPUT "NOMBRE: ";A$
50 PRINT UPPER$(A$),LOWER$(A$)
60 NEXT
```

Fig. 8.11 Conversión de mayúsculas a minúsculas, y viceversa, con LOWER$ y UPPER$.

cadena en mayúsculas; LOWER$ las convierte en minúsculas. Esto es útil cuando hay que manejar palabras mixtas, que puedan estar en mayúsculas o en minúsculas. Por ejemplo, cuando se ordenan cadenas por orden alfabético. Como ya hemos visto, al comparar las cadenas el ordenador traduce los caracteres a códigos ASCII. Los códigos de las minúsculas son mayores que los de las mayúsculas. Así, cuando la máquina ordena las palabras CODO, cada, BUENO y bien, las deja en el orden CODO, BUENO, cada, bien. Si en el proceso de comparación se cambian provisionalmente todas las palabras a mayúsculas o a minúsculas, el orden que se obtiene sí es el correcto. Dicho sea de paso, y ya que estamos hablando de ordenación, hay una instrucción muy útil: FRE. Su funcionamiento es el siguiente: PRINT FRE(0) escribe el espacio de memoria disponible (en bytes); PRINT FRE ("") hace algo similar, pero reorganizando antes la zona de memoria en la que se almacenan las cadenas literales. Esta reorganización es importante porque, cuando se ha estado ordenando cadenas, en la memoria queda un revoltijo de espacios inútiles en los que el ordenador ha almacenado las cadenas que tenía que intercambiar. FRE ("") libera ese espacio; utilizando esta instrucción durante la ejecución de un programa largo de ordenación de cadenas se mejora notablemente la velocidad y eficacia del programa.

¿Quiere escribir en blanco? Hay una función literal SPACE$, que se lo facilita. Por ejemplo S$ = SPACE$(40) es una cadena que consta de 40 espacios. Al ejecutar PRINT S$ se escribe una línea en blanco, lo que no deja de ser más fácil que tener que programar un bucle para escribir el espacio 40 veces. Otra instrucción interesante: WRITE. Si se programa WRITE «PAPEL» en lugar de PRINT «PAPEL», el ordenador escribe la palabra *y también* las comillas. También se puede definir A$ = "PAPEL" y luego hacer WRITE A$. Escribir una frase entre comillas de esta forma es sencillo; con instrucciones PRINT es bastante más difícil.

Y vamos con la última. Las teclas del CPC464 son de repetición. Para la mayor parte de las aplicaciones, la velocidad de repetición es perfecta, pero puede ser necesario cambiarla. Por ejemplo, para algunos juegos esa velocidad puede ser demasiado baja, y para programas de proceso de datos demasiado alta. En algunos casos se deseará una velocidad tan baja que las teclas no repitan casi nunca. Por ejemplo, si el programa tiene dos instrucciones INKEY$ seguidas y el usuario mantiene pulsada durante demasiado tiempo la tecla de la primera respuesta, pudiera ocurrir que el segundo INKEY$ le pasara desapercibido, lo que podría dar como resultado que se ejecutara indebidamente una opción de un menú. La acción de las teclas se con-

trola mediante la instrucción SPEED KEY seguida de dos parámetros. El primero especifica durante cuánto tiempo tiene que estar pulsada una tecla antes de que empiece la repetición. El segundo especifica el intervalo entre repeticiones. El margen para estos números es de 0 a 255. Si los números son demasiado pequeños, el teclado puede ser incontrolable, por ejemplo, por la aparición de varias letras cuando pulse una tecla una sola vez. Escriba primero SPEED KEY 2,2 y luego, si puede, SPEED KEY 255,255. En la primera velocidad es casi imposible escribir; la segunda prácticamente inhibe la repetición.

9

Introducción a los gráficos

A todo ordenador moderno se le pide que maneje los colores de forma espectacular y que sea capaz de realizar otros efectos especiales. El CPC464 no deja nada que desear en este aspecto. En el presente capítulo vamos a comenzar el estudio de las posibilidades gráficas de esta máquina. Para empezar tenemos que conocer la terminología. El primer término, *gráficos*, se refiere a los dibujos y formas que se pueden dibujar en la pantalla; todos los ordenadores modernos están dotados de instrucciones que permiten dibujar tales formas. Los gráficos pueden ser de «baja resolución» o de «alta resolución». El término *resolución* no es fácil de explicar.

Supongamos que tenemos una hoja de papel y que queremos formar imágenes en ella. Cuadriculamos la hoja dividéndola en 936 rectángulos y formamos las imágenes pintando cada uno de los rectángulos de un solo color. Fácilmente se comprende que las imágenes resultantes no podrán tener mucho detalle. Pues bien, algunos ordenadores no dan mejores resultados. Supongamos, en cambio, que pudiéramos cuadricular el papel en 32 000 rectángulos: la nitidez de las imágenes podría ser muchísimo mejor; en este caso estaríamos trabajando «en alta resolución». En el CPC464 se puede elegir la resolución más adecuada a cada trabajo. El número de 32 000 puntos corresponde al modo de baja resolución del CPC464, el modo 0. El modo de alta resolución controla 128 000 puntos distintos de la pantalla, si bien solamente en dos colores. Para controlar imágenes tan detalladas, el ordenador necesita gran espacio de memoria. Muchos ordenadores tienen compartida la memoria de la pantalla con la que el usuario necesita para sus programas. Así, hay ordenadores de los que se dice que son de 64K o 32K y sin embargo para el usuario queda muy poca memoria libre, en ocasiones no más de 6K. El resto de la memoria está dedicado a la pantalla y a otras funciones del sistema operativo. El caso del CPC464 es distinto. Reserva 16K de memoria permanentemente para la pantalla, pero respeta los restantes 43533 bytes (42.5K), que quedan a disposición del usuario.

Hasta ahora sólo hemos trabajado con la «pantalla de textos». En esta pantalla solamente podemos visualizar textos, esto es, caracteres que se pueden enviar a ella mediante instrucciones PRINT. Más adelante veremos que hay instrucciones que permiten acceder a las «pantallas gráficas».

GRÁFICOS DE TECLADO

Algunas formas gráficas, las ilustradas en la figura 9.1, se obtienen directamente pulsado diversas teclas junto con CTRL. Sin embargo, estos caracteres no se pueden escribir por los mismos métodos que los caracteres normales, es decir, mediante instrucciones PRINT seguidas de un signo entre comillas. Para escribirlos hay que recurrir a los códigos de caracteres.

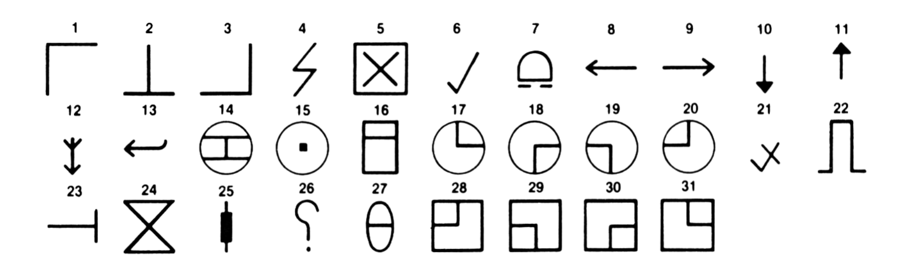


Fig. 9.1 Caracteres gráficos que se obtienen directamente del teclado, junto con sus códigos ASCII.

CÓDIGOS DE CARACTERES

El método, como decimos, es utilizar los códigos ASCII. En realidad, hay dos grupos de códigos ASCII que producen formas gráficas. Uno de ellos es el de los códigos ASCII 128 a 255. Puede verlos en el apéndice III del manual, pero no obstante se lo recordamos con el programa de la figura 9.2. En este grupo hay letras griegas

```
10 CLS:N=127
20 FOR J=1 TO 128
30 PRINT CHR$(N+J);" ";
40 IF J/20=INT(J/20) THEN PRINT
50 NEXT
```

Fig. 9.2 Programa que escribe en la pantalla los caracteres correspondientes a los códigos 128 a 255.

que son útiles en aplicaciones técnicas y científicas. El programa escribe los caracteres separados unos de otros; párese a pensar cómo lo consigue. El otro grupo de caracteres gráficos es el de códigos ASCII 1 A 31. La dificultad es que estos mismos códigos se utilizan también con otros propósitos, y es necesario saber cómo cambiar de una función a otra. La clave está en CHR$(1). Para visualizar los signos

hay que enviar esos códigos a la pantalla precedidos de CHR$(1). Por ejemplo, PRINT CHR$(8) mueve el cursor hacia la izquierda; en cambio, PRINT CHR$(1) + CHR$(8) escribe el signo. El programa de la figura 9.3 escribe esos 31 signos, algunos de los cuales son útiles en aplicaciones comerciales y educacionales, no sólo en juegos.

```
10 CLS
20 FOR N=1 TO 31
30 PRINT N;" ";CHR$(1)+CHR$(N);" ";
40 IF N/5=INT(N/5)THEN PRINT:PRINT
50 NEXT
```

Fig. 9.3 Programa que visualiza los caracteres gráficos de códigos comprendidos entre el 1 y el 31. Obsérvese que PRINT CHR$(X), con X entre 1 y 31, no escribe estos caracteres.

Estos caracteres se pueden utilizar con efectos ornamentales utilizando las plantillas que se facilitan en el manual (apéndice VI). La plantilla para el modo 1 es de 40 columnas y 25 filas, pues ésta es la capacidad de la pantalla en ese modo. En modo 0 los caracteres son el doble de anchos, y sólo caben 20 por línea. En modo 2 hay 80 caracteres por línea. El número de filas es en todos los casos el mismo: 25; esto implica que la altura de los caracteres es siempre la misma. Cada cuadrado de la plantilla representa la posición de un carácter. El procedimiento normal, para no

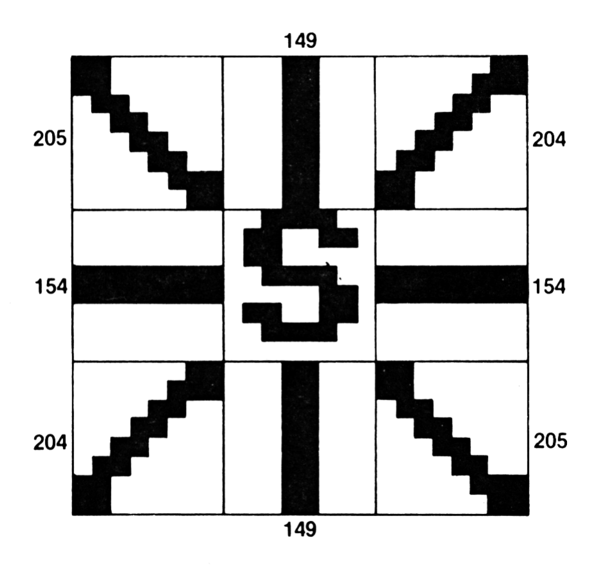


Fig. 9.4 Diseño de un logotipo con caracteres gráficos.

estropear la plantilla, consiste en poner sobre ella un papel vegetal. Dibujando sobre este papel nos podemos hacer una idea de cómo resultará la forma diseñada en la pantalla. Para programar un diseño de este tipo hay tres métodos. Uno consiste en escribir cada una de las líneas de la pantalla por separado; el otro, en construir un bucle que lea los códigos de los caracteres de una línea DATA y los envíe a la pantalla. El tercero consiste en poner todos los caracteres en una cadena literal, igual que si se tratase de palabras.

En la figura 9.4 damos el diseño de un logotipo, como podría ser el de una marca comercial. Consiste en una letra «S» sobre cuatro rayas dispuestas radialmente. La letra es lo más sencillo: CHR$(83). Los demás caracteres han sido elegidos del conjunto que se ilustra en el manual. Ahora lo que falta es escribir el programa que

```
5 PAPER 3:PEN 2
10 CLS:PRINT:PRINT
20 PRINT TAB(5) CHR$(205)+CHR$(149)+CHR$(204)
30 PRINT TAB(5) CHR$(154)+CHR$(83)+CHR$(154)
40 PRINT TAB(5) CHR$(204)+CHR$(149)+CHR$(205)
50 PRINT:PRINT
```

Fig. 9.5 Programa que visualiza en la pantalla el logotipo de la figura 9.4.

```
10 PAPER 3:PEN 2:CLS
20 FOR J%=1 TO 3:LOCATE 18,J%+10
30 FOR N%=1 TO 3
40 READ D%:PRINT CHR$(D%);
50 NEXT:PRINT:NEXT
60 WINDOW#2,1,40,24,25
70 WINDOW SWAP 0,2
100 DATA 205,149,204
110 DATA 154,83,154
120 DATA 204,149,205
```

Fig. 9.6 Lectura de los caracteres gráficos mediante bucles.

envíe estos caracteres a la pantalla. El programa de la figura 9.5 lo hace, pero esta forma de programar es bastante torpe. Observemos entonces el programa de la figura 9.6. Aparentemente no es una gran mejora, e incluso tiene más líneas, pero es un programa mucho mejor. Aquí ya sólo hay una instrucción PRINT, no tres. Utiliza dos bucles, uno para las filas y otro para las columnas. Para que la ejecución sea más rápida, todas las variables son enteras (seguidas del signo %). Además, la

posición del logotipo está controlada por la instrucción LOCATE, dentro del bucle. La ventaja de este método es que los datos son más visibles, y más fácilmente modificables sin alterar el resto del programa. Obsérvese que hemos definido una ventana, que luego intercambiamos con el número 0 para que el mensaje «Ready» y el cursor queden al fondo de la pantalla, no inmediatamente debajo del logotipo. Después de ejecutar este teclado habrá que teclear otra orden WINDOW SWAP 0,2 para volver a disponer de la pantalla completa.

```
10 PAPER 2:PEN 3:CLS
20 GR$="":B$=STRING$(3,8)+CHR$(10)
30 FOR J%=1 TO 3:FOR N%=1 TO 3
40 READ D%:GR$=GR$+CHR$(D%):NEXT
50 GR$=GR$+B$:NEXT
60 LOCATE 18,12:PRINT GR$
70 DATA 205,149,204,154,83,154,204,149,205
```

Fig. 9.7 Asignación de los caracteres gráficos a una cadena, que luego se puede escribir con una sola instrucción.

El programa de la figura 9.7 es aun mejor. Empieza por definir la cadena B$. Sus tres primeros caracteres son retrocesos del cursor (8), seguidos del 10, que es la bajada del cursor a la línea siguiente. El efecto de escribir B$ es, pues, mover el cursor tres posiciones hacia la izquierda y una hacia abajo. En la línea 20 se iguala la cadena GR$ a una cadena vacía. A continuación se abren dos bucles, uno para las filas y el otro para las columnas. Cada pasada del bucle externo lee una línea de datos, los coloca al final de GR$ y luego une GR$ con B$ (movimiento del cursor). El efecto global de una línea es escribir los tres caracteres y colocar el cursor al principio de la línea siguiente.

La ventaja principal del programa de la figura 9.7 es que permite escribir el logotipo en cualquier lugar de la pantalla sin requerir ninguna modificación del programa. Siempre que se ejecute PRINT GR$ el dibujo aparecerá en el lugar en que se encuentre el cursor en ese momento. Desde luego, hay que evitar que el cursor esté demasiado cerca del borde derecho o del inferior de la pantalla.

Ahora que ya sabemos generar formas, vamos a ver cómo podemos dotarlas de cierta animación. Consideremos el ejemplo de la figura 9.8. La idea consiste en escribir la forma en un determinado lugar de la pantalla, especificado por LOCATE. Se mantiene la forma en la pantalla durante algún tiempo y luego se la borra. A continuación se modifica la posición con otra instrucción LOCATE y se repite el proceso. Si el retardo es pequeño, como en este ejemplo, y el desplazamiento es pequeño, y en este ejemplo *no* lo es, la ilusión de movimiento es bastante buena. Pero la animación de movimientos no es la especialidad de la pantalla de texto, pues la distancia entre caracteres es demasiado grande.

```
10 PAPER 2:PEN 3:CLS
20 GR$="":B$=STRING$(3,8)+CHR$(10)
30 FOR J%=1 TO 3:FOR N%=1 TO 3
40 READ D%:GR$=GR$+CHR$(D%):NEXT
50 GR$=GR$+B$:NEXT
60 A$=STRING$(3,32):SP$="":SP$=A$+B$+A$+B$+A$
70 FOR N=1 TO 18
80 LOCATE N,12
90 PRINT GR$:GOSUB 1000
100 LOCATE N,12
110 PRINT SP$:GOSUB 1000
120 NEXT:LOCATE N,12:PRINT GR$
130 DATA 205,149,204,154,83,154,204,149,205
140 END
1000 FOR K=1 TO 20:NEXT:RETURN
```

Fig. 9.8 Ejemplo sencillo de movimiento en la pantalla de texto.

DEFINICIÓN DE CARACTERES

El CPC464 ofrece una interesante posibilidad de producir gráficos con movimiento en la pantalla de texto utilizando solamente instrucciones PRINT para colocar las formas. Estos gráficos se podrían denominar «de baja resolución», en el sentido que el número de posiciones de la pantalla en los que se los puede situar es bastante limitado. El título de esta sección da idea de lo que nos proponemos: crear nuestros propios caracteres a medida. En este proceso hay dos etapas: el diseño de las formas y su colocación en la pantalla.

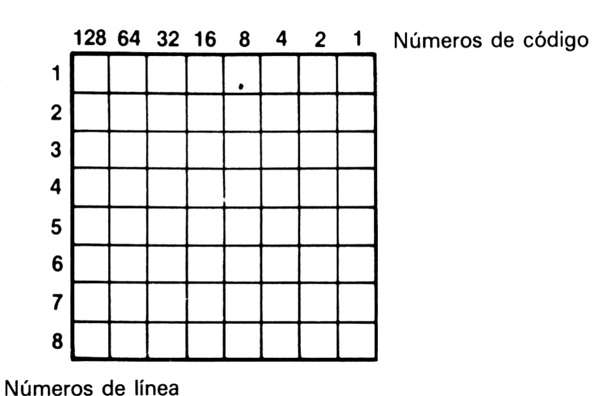


Fig. 9.9 Retícula para la definición de caracteres.

Como es natural, empezaremos por el diseño. El tamaño de la forma que vamos a diseñar es el de un carácter de texto o, dicho de otro modo, el del bloque del cursor. Este carácter, como todos los demás del CPC464, está formado por 64 puntos, dispuestos en una matriz o retícula de 8 filas y 8 columnas. En la figura 9.9 hemos dibujado esa retícula. Usted puede reproducirla en un papel cuadriculado o por fotocopia. La cuestión es que cada uno de los puntos de la retícula representa un punto de la pantalla, que puede ser de color del papel (PAPER) o del de la pluma (PEN), dependiendo del valor de unos códigos que tenemos que suministrar al ordenador. Cuando se diseña una letra en una retícula 8 × 8, normalmente sólo se aprovechan 7 filas y 7 columnas, de forma que se deja libre la columna de la derecha y la fila inferior para que haya alguna separación entre dos caracteres seguidos y entre dos líneas de texto. En cambio, al diseñar formas gráficas puede interesar ocupar todo el bloque para aprovechar los efectos que pueda tener unir unas formas con otras.

El manual del CPC464 explica muy someramente cómo diseñar estas formas, y es posible que el lector haya quedado confuso. En primer lugar hay que entender el significado de los números que aparecen en la cabecera de las ocho columnas de la figura 9.9. Cada número es un código para los cuadrados de la figura correspondiente. La presencia de ese número significa que el color del cuadrado es el de la pluma; su ausencia (es decir, poner un 0 en su lugar) significa que el color del cuadrado es el del papel. Veamos el ejemplo de la figura 9.10.

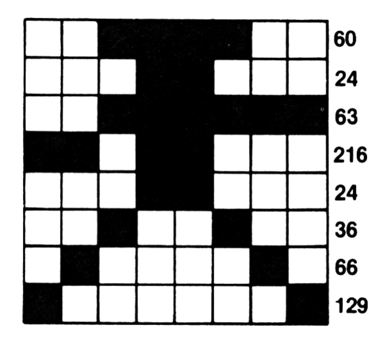


Fig. 9.10 Cómo utilizar la retícula para dibujar un «astronauta».

La primera línea tiene cuatro cuadrados negros. Esos cuatro cuadrados son los que deben aparecer en el color de la pluma; sus números de código son 32, 16, 8 y 4. La suma de estos cuatros números es 60, y este número lo anotamos al lado de la línea. Análogamente, los códigos de los dos cuadrados negros de la segunda línea son 16 y 8; anotamos su suma, 24, al lado. Continuamos de esta manera hasta completar las 8 líneas.

Hagamos algunas observaciones. La primera es que el código de una línea en blanco es cero. Otra es que se ahorran bastantes cálculos si se tiene en cuenta que para una línea completamente negra la suma es 255. Cualquiera que sea la forma que se esté diseñando, al final siempre tenemos que terminar con ocho números.

Ahora tenemos que decirle al ordenador cómo es la forma que hemos diseñado. En el fondo, lo que haremos es almacenar en la memoria los ochos códigos que hemos calculado junto con el código ASCII que queremos que tenga nuestro diseño. Hay, cómo no, una instrucción que se encarga de esto: SYMBOL. Esta instrucción va seguida de *nueve* parámetros. El primero es el código ASCII al que queremos asignar nuestra forma; los otros ochos números son los códigos que calculamos antes. Todos los números van separados por comas; entre la L y el primer número se deja un espacio; el orden de los códigos es de arriba abajo.

¿Qué códigos ASCII podemos utilizar? Sin más preparativos, los comprendidos entre el 240 y el 255, 16 en total. Doce de ellos son generados normalmente por las teclas de movimiento del cursor (son doce porque junto con esas teclas se pueden pulsar también las teclas SHIF y CTRL). Pero esto no quiere decir que si se redefinen esos caracteres las teclas funcionen de otra manera: al pulsar una tecla de cursor no va a aparecer un astronauta en la pantalla. Supongamos que queremos asignar nuestro diseño al código 244. El programa necesario es el de la figura 9.11. Una vez ejecutado este programa, podemos escribir la nueva forma donde queramos utilizando PRINT CHR$(244).

```
10 CLS
20 SYMBOL 244,60,24,63,216,24,36,66,129
30 PRINT:PRINT TAB(12)CHR$(244)
```

Fig. 9.11 Visualización del astronauta.

Como ya hemos sugerido, no nos vemos obligados a elegir uno de los códigos del 240 al 255 para este fin. En realidad, podemos redefinir *todos* los caracteres del 32 en adelante. Veamos el ejemplo de la figura 9.12. En la línea 10 hemos puesto una instrucción nueva: SYMBOL AFTER 90. Esto nos permite redefinir cualquier carácter cuyo código ASCII sea 90 o superior. Hemos elegido el 92, que es el código de la barra inclinada hacia la izquierda (tecla que está al lado de la de SHIFT derecha).

```
10 CLS:SYMBOL AFTER 90
20 SYMBOL 92,60,24,63,216,24,36,66,129
30 PRINT:PRINT TAB(12);"\"
```

Fig. 9.12 Redefinición del carácter generado por una tecla.

Cuando ejecute este programa verá que en la pantalla aparece el astronauta en respuesta a la instrucción PRINT TAB (12);"\". (También podríamos haber puesto CHR$(92); como prefiera.) Más importante aún; el astronauta saldrá en la pantalla cada vez que pulse la tecla. Para dejar las cosas como estaban, grabe el programa y pulse SHIFT CTRL ESC para reinicializar la máquina.

Observe que los demás códigos posteriores al 89 no han resultado afectados; sólo el que hemos redefinido. Nada nos impide redefinir todos los caracteres, pero esto es una tarea muy laboriosa. Aunque solo sea como curiosidad, teclee y ejecute el programa de la figura 9.13, que redefine todos los caracteres. Verá que todo lo que ve en la pantalla, salvo el cursor, tiene la forma del astronauta, incluso los mensajes tales como «Ready» y «Syntax error». Ésta es una buena forma de desanimar a los curiosos que quieran ponerse a jugar con su ordenador cuando usted no está presente.

```
10 CLS:SYMBOL AFTER 32
15 FOR N=32 TO 127
20 SYMBOL N,60,24,63,216,24,36,66,129
25 NEXT
```

Fig. 9.13 Redefinición de todas las teclas. Después de ejecutar este programa le será difícil seguir trabajando. Grábelo y reinicialice la máquina pulsando CTRL SHIT ESC.

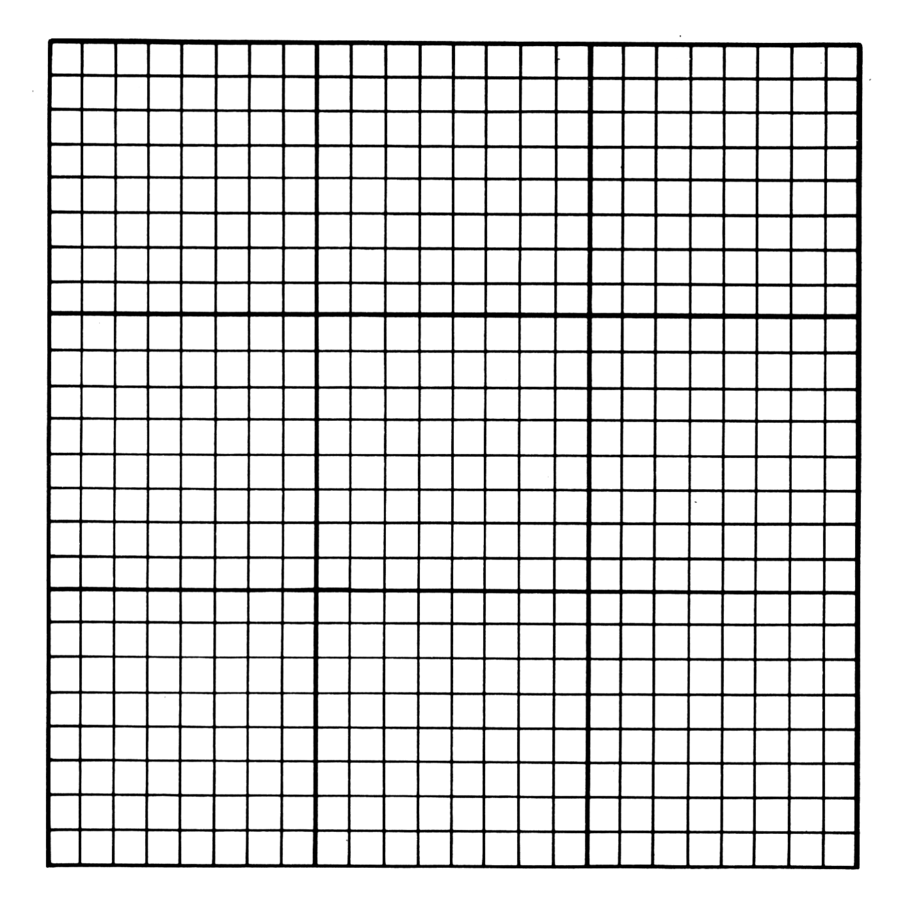


Fig. 9.14 Retícula para el diseño de 9 caracteres adyacentes.

Tendrá que pulsar SHIFT CTRL ESC para volver al funcionamiento normal del teclado, pero recuerde que esto borrará el programa.

Tampoco hay por qué limitarse a definir un solo carácter. Se pueden definir varios y combinarlos para dar una forma más compleja. En la figura 9.14 se da una retícula para el diseño de un bloque de 9 caracteres. Supongamos que asignáramos tres nuevos caracteres a los códigos 240 a 242. Podríamos entonces ponerlos los tres juntos en una cadena, GR$, para luego escribirlos a la vez con PRINT GR$. En la figura 9.15 se puede ver un ejemplo. El programa de la figura 9.16 los pone en movimiento. El lector ya no necesita que se lo expliquemos con mucho detalle. Hemos utilizado los códigos 240 a 242, asignándoles diseños nuevos. Como hemos ocupado

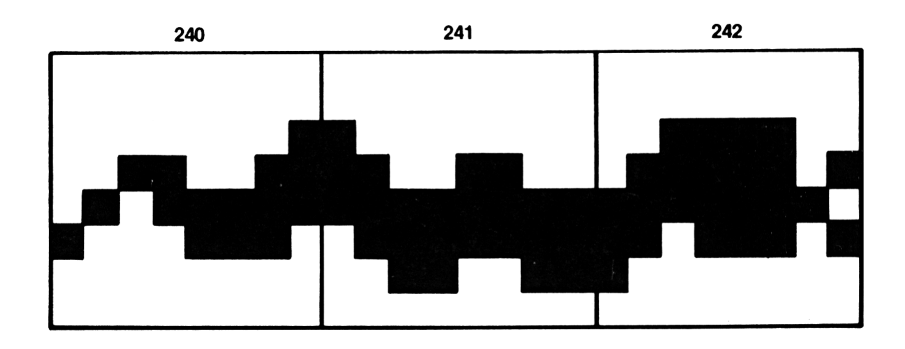


Fig. 9.15 Diseño de una serpiente con tres caracteres.

```
10 CLS
20 SYMBOL 240,0,0,1,51,95,142,0,0
30 SYMBOL 241,0,0,128,204,255,127,51,0
40 SYMBOL 242,0,0,60,125,254,221,128,0
50 GR$=CHR$(240)+CHR$(241)+CHR$(242)
60 X=2:Y=12
70 EVERY 10 GOSUB 1000
80 FOR N=1 TO 25:LOCATE 5,N:PRINT"HISSIN G SID"
90 FOR J=1 TO 1000:NEXT:NEXT
100 GOTO 100
110 END
1000 LOCATE X-1,Y:PRINT" ":LOCATE X,Y
1010 PRINT GR$;
1020 X=X+1
1030 IF X>38THEN LOCATE X-1,Y: PRINTCHR$(18);:X=2
1040 RETURN
```

Fig. 9.16 Animación de la serpiente utilizando EVERY.

toda la anchura de las retículas 8 × 8, las tres piezas de la serpiente quedan conectadas. ¿No es fascinante?

Y así hemos llegado a una nueva instrucción, EVERY, muy útil en los programas de animación. Estudiaremos esta instrucción, y otras de su género, más adelante. Su interés reside en que permite acceder a los cuatro cronómetros internos del CPC464. La idea es hacer algo con una frecuencia especificada. La frecuencia depende del número que siga a EVERY. El número 50 significa una vez por segundo. Se puede hacer, por ejemplo, que con la frecuencia deseada se ejecute cierta subrutina. Llegado el momento, el ordenador aplaza lo que esté haciendo y se pone a ejecutar la subrutina; cuando termina continúa con la tarea que había aplazado. Se aparenta así que el ordendor está haciendo dos cosas al mismo tiempo. Observe en el ejemplo que mientras se ejecuta el bucle que escribe «LINA LA VIPERINA» la serpiente se mueve. Cuando este bucle termina, empieza un bucle infinito en la línea 100. Este bucle mantiene el programa, y por lo tanto el cronómetro, en marcha. El cronómetro funciona independientemente del resto del programa, y puede interrumpir las demás acciones para ejecutar su rutina cuando haya transcurrido el tiempo especificado. En este ejemplo el intervalo es 10, o sea, un quinto de segundo. Una forma que pueda ser movida independientemente del resto del programa es lo que se denomina *agente móvil* o *sprite*. Muchas máquinas modernas permiten el uso de agentes móviles, pero pocas con la sencillez y posibilidades de control del CPC464.

Puesto que hay cuatro cronómetros, se pueden controlar cuatro grupos independientes de agentes móviles; cada cronómetro puede controlar cuantas acciones simultáneas se desee. El cronómetro que se deba utilizar se especifica por su número, del 0 al 3, puesto a continuación del intervalo. Por ejemplo, EVERY 10,0 GOSUB 1000 y EVERY 20,3 GOSUB 2000.

```
10 CLS
20 FOR Y=1 TO 25
30 PRINT TAB(RND(1)*35);POS(#0);",";VPOS(#0)
40 FOR N=1 TO 1000:NEXT
50 NEXT
```

Fig. 9.17 Utilización de POS y VPOS para determinar la posición del cursor.

Otra posibilidad del CPC464 es determinar la posición del cursor. Las instrucciones son POS y VPOS, seguidas necesariamente de un número de ventana. Para la pantalla completa ya sabemos que el número es el 0. En la figura 9.17 se da un ejemplo de utilización de estas instrucciones. POS da la posición horizontal del cursor, y VPOS la vertical. Los números tienen el mismo significado que en la instrucción LOCATE. POS y VPOS son útiles en particular cuando se trabaja con ventanas, porque no siempre es fácil seguirle la pista a la posición del cursor.

10

Gráficos más avanzados

La capacidad gráfica del CPC464 es muy superior a la de muchas máquinas de su categoría. En este capítulo vamos a estudiar las instrucciones específicas de los gráficos, es decir, aquéllas que crean imágenes sin utilizar PRINT ni CHR$. Lo primero que salta a la vista en este ordenador es la gran resolución de sus gráficos. Si observa muy de cerca la pantalla de un televisor de color verá que está dividida en líneas de puntos. Estos puntos son los que emiten la luz, de modo que ninguna imagen puede ser menor que la anchura de un punto o que la distancia de una línea o otra. Los «puntos» con los que trabaja un ordenador son los denominados *pixels* (de *picture elements*, elementos de imagen). Cada pixel corresponde en tamaño a varios puntos de un televisor. El CPC464 ofrece tres resoluciones distintas, dependientes del modo de pantalla elegido. En modo 0, la resolución es 160 pixels en horizontal y 200 en vertical. En los otros modos la resolución vertical sigue siendo la misma. En horizontal es 320 pixels en modo 1 y 640 en modo 2. Esta máxima resolución no se aprecia en un televisor; solamente es aprovechable en un monitor. Tan gran resolución tiene un precio, que es la limitación del número de colores: solamente dos.

La posibilidad de «pintar» cada uno de los 64 000 puntos (modo 1) no sería útil en absoluto si para crear las imágenes hubiera que especificar la posición y el color de cada punto uno por uno. Hay ordenadores así de torpes, pero el CPC464, como el lector ya habrá supuesto, es bastante más hábil. De hecho, hay gran variedad de instrucciones gráficas que permiten dibujar de forma más eficaz. Todas estas instrucciones utilizan un sistema de coordenadas X, Y similar al de la instrucción LOCATE, con la diferencia de que ahora los números son distintos. En todos los modos gráficos la coordenada X varía entre 0 y 199. Esto es afortunado porque si se ha escrito un programa para un modo y se lo quiere ejecutar en otro, no hay que recorrer el programa para modificar todos los números. En la figura 10.1 se explica cómo dibujar una plantilla para gráficos.

---

1. Tome una hoja de papel milimetrado de formato A4.
2. Escriba los números en los ejes, de 25 en 25, hasta 600 en uno y hasta el 400 en el otro.
3. Haga sus dibujos en papel vegetal superpuesto a esta plantilla.

Fig. 10.1 Preparación de una plantilla para gráficos para utilizar con papel vegetal.

Hay diferencias notables entre trabajar con texto y hacerlo con gráficos. Las instrucciones gráficas actúan directamente sobre los puntos, de forma que las unidades de X e Y son mucho más pequeñas. Además, el origen del sistema de coordenadas en los gráficos es distinto. El origen es el punto de coordenadas X = 0 e Y = 0. En la pantalla de texto no se puede utilizar el cero, pero LOCATE 1,1 es el extremo superior izquierdo de la pantalla. En cambio, en la pantalla gráfica el punto 0,0 (es decir, X = 0, Y = 0) está en el extremo *inferior* izquierdo. Las coordenadas Y crecen *hacia arriba*, y las X de izquierda a derecha. La situación de este origen es la habitual en los diversos sistemas de representación gráfica. Si el lector está familiarizado con ellos no le será difícil habituarse a las instrucciones gráficas de este ordenador. Vamos a empezar por la instrucción PLOT.

PLOT

La representación gráfica es una de las aplicaciones más importantes de la capacidad gráfica de un ordenador si éste ha ser utilizado con programas comerciales o educacionales. Una gráfica (en femenino) es un conjunto de puntos que pueden ser unidos entre sí y que contienen alguna información. El CPC464 dibuja puntos de gráficas mediante las instrucciones PLOT y PLOTR. El color del punto dibujado será el de la pluma actual. La diferencia entre estas dos instrucciones es la siguiente: PLOT utiliza coordenadas absolutas; PLOTR utiliza coordenadas *relativas a la posición actual del cursor*. Por ejemplo, PLOT 0,0 dibuja un punto en el origen de coordenadas, mientras que PLOTR 0,0 lo dibuja en la posición del cursor, cualquiera que ésta sea. En las instrucciones gráficas el cursor no es el bloque que hemos conocido en la pantalla de texto; en gráficos es invisible, y su posición no se evidencia mientras no dibuje un punto con una tinta distinta de la del papel. El cursor ordinario vuelve a aparecer cuando concluye la ejecución del programa gráfico.

```
10 MODE 0
20 INK 0,0
30 FOR X=1 TO 639
40 Y=200+200*SIN(2*PI*X/639)
50 PLOT X,Y,1
60 Y=200+200*(SIN(2*PI*X/639)↑2)
70 PLOT X,Y,2
80 Y=200+200*(SIN(2*PI*X/639)↑3)
90 PLOT X,Y,3
100 NEXT
110 GOTO 110
```

Fig. 10.2 Programa de representación gráfica en alta resolución. Este programa dibuja tres curvas con colores diferentes.

El programa 10.2 dibuja tres gráficas al mismo tiempo, y a velocidad considerable. En la línea 30 se establece el margen de X de forma que cubra toda la pantalla. Para cada valor de X se calculan tres valores de Y. El primero se obtiene del seno del ángulo cuyo valor es proporcional a X/639; el segundo, del cuadrado de ese seno; y el tercero, del cubo. Los tres puntos se dibujan en las instrucciones 40, 60 y 80, con tres colores de tinta diferentes. El número 200 que interviene en las fórmulas de Y tiene como objeto cubrir la pantalla con las gráficas. El seno de un ángulo tiene un valor que está comprendido entre −1 y +1. Pero un desplazamiento de una unidad en el eje Y no es muy impresionante, y por eso lo multiplicamos por 200. El margen de variación se convierte así en −200, +200. Ahora bien, en esta pantalla no podemos dibujar un punto cuya coordenada Y sea −200; sumando 200 a todos los valores de Y hacemos que el margen vaya de 0 a 400, que coincide con el margen de variación en el eje vertical de la pantalla. Otro detalle que se debe observar es la expresión que hemos puesto para el ángulo: 2 * PI * X/639. Esto es necesario para que el ángulo quede en radianes. Los ángulos en las funciones trigonométricas varían entre 0 y 2 * PI radianes. Cuando X es 0, 2 * PI * X/639 es también cero, lo que da cero radianes en el extremo izquierdo de la gráfica. Cuando X es 639, 2 * PI * 639 es 2 * PI radianes, precisamente lo que queríamos. Si no le resulta cómodo trabajar con radianes, ponga la instrucción DEG en alguna línea anterior a la primera en que aparezcan funciones trigonométricas. Cuando se ejecute DEG el ordenador entenderá que todos los ángulos están en grados. Para volver a radianes, ejecute RAD o reinicialice la máquina.

Ahora ejecute el programa y observe las gráficas. Aquí se puede apreciar perfectamente el tamaño de un punto en modo 0. También se observa que la «baja resolución» del CPC464 es bastante mejor que la «alta resolución» de otros ordenadores. Cambie la línea 10, primero a MODE 1 y luego a MODE 2; observe cómo el tamaño de los puntos se va reduciendo a medida que aumenta la resolución. Los colores son más difíciles de apreciar, pero las gráficas son más «suaves» que en modo 0. Cuando ejecute el programa en modo 2 solamente verá dos gráficas, ya que en este modo sólo disponemos de dos colores. En la mayor parte de los casos el modo 1 es una buena solución intermedia en cuanto a número de colores y resolución.

Pero todavía no hemos terminado con el tema de la representación gráfica. Pruebe la modificación del programa anterior que se muestra en la figura 10.3. El efecto de la instrucción ORIGIN 0,200 es trasladar el origen del sistema de coordenadas. Al desplazar la posición del origen evitamos tener que sumar 200 en todos los cálculos de Y. Ahora ya podemos hacer cosas como PLOT 50, −50, porque −50 significa «50 puntos por debajo del origen». La instrucción ORIGIN se utiliza también para especificar el tamaño de una ventana, según se explica en el manual.

DIBUJO DE RECTAS

La representación gráfica es una de las posibilidades de los gráficos de alta resolución. Pero hay instrucciones que nos permiten hacer muchas otras cosas. Dos de ellas son MOVE y DRAW. La instrucción MOVE mueve el cursor gráfico. Su efecto no es inmediatamente evidente; es como mover el pincel sin tocar el papel. DRAW

```
10 MODE 0:ORIGIN 0,200
20 INK 0,0
30 FOR X=1 TO 639
40 Y=200*SIN(2*PI*X/639)
50 PLOT X,Y,1
60 Y=200*(SIN(2*PI*X/639)↑2)
70 PLOT X,Y,2
80 Y=200*(SIN(2*PI*X/639)↑3)
90 PLOT X,Y,3
100 NEXT
110 GOTO 110
```

Fig. 10.3 Redefinición del origen para desplazar las gráficas. Ésta es la forma más sencilla de utilizar la instrucción ORIGIN.

```
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAW 150,100
50 DRAW 450,300
60 DRAW 450,100
70 DRAW 150,100
80 GOTO 80
1000 MODE 1
1010 LIST
```

Fig. 10.4 Dibujo de un cuadrado con la instrucción DRAW.

mueve el cursor gráfico y dibuja una recta al mismo tiempo. Las dos instrucciones utilizan el mismo sistema de coordenadas que hemos visto antes. DRAW puede utilizar un tercer parámetro: el número del tintero.

Veamos el ejemplo de la figura 10.4. Este programa no es nada ambicioso; no hace más que dibujar un cuadrado. El color de las rayas será dorado, sobre fondo azul, si acaba de encender la máquina; pero seguramente será rojo sobre negro si acaba de ejecutar el programa de la figura 10.3. La razón es que, si no se especifica un número de tintero en la instrucción DRAW X,Y, el color con que se dibuja es el último especificado en una instrucción DRAW o PLOT. Observe también que el dibujo se realiza muy deprisa, y compare esta velocidad con la de otros ordenadores.

En este programa hay una instrucción nueva: ON BREAK GOSUB 1000 (en caso de interrupción ir a ...). Su efecto consiste en que si se pulsa dos veces la tecla ESC,

en lugar de detenerse el programa se ejecuta la rutina que empieza en la línea 1000. La razón por la que hemos puesto aquí esta instrucciöon es que los listados en modo 0 son casi ilegibles; en caso de interrupción, la subrutina de 1000 convierte la pantalla a modo 1 y lista el programa. Puesto que la instrucción LIST siempre termina el programa, no tiene objeto poner RETURN al final de esta «subrutina». Así pues, cuando se pulsa ESC dos veces para interrumpir el bucle infinito de la línea 80, la pantalla vuelve a modo 1 y se lista el programa.

Existe también la versión de DRAW que utiliza coordenadas relativas. Por ejemplo, DRAWR 10,100 dibuja una recta desde la posición actual del cursor hasta el punto que se encuentra a 10 pixels a la derecha y 100 pixels por encima de ella. Esta acción no es la misma que dibujar una recta que vaya al punto 10,100, que es lo que haría DRAW. El programa de la figura 10.5 dibuja el cuadrado de antes con DRAWR. Las distancias son números positivos si se desea mover el cursor hacia arriba o hacia la derecha, y negativos para los sentidos contrarios. Si una de las coordenadas, X o Y, ha de permanecer constante, el número que hay que poner es el 0.

```
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAWR 0,200,2
50 DRAWR 300,0,2
60 DRAWR 0,-200
70 DRAWR -300,0
80 GOTO 80
1000 MODE 1
1010 LIST
```

Fig. 10.5 Dibujo de un cuadrado con la instrucción DRAWR. Nótese la diferencia en los parámetros.

Para terminar de dejar clara la diferencia entre DRAW y DRAWR consideremos el programa de la figura 10.6. Este programa dibuja un conjunto de rectas trazadas desde el centro de la pantalla hasta puntos determinados aleatoriamente. Tras una breve pausa, de dibuja otro conjunto de rectas, pero esta vez con DRAWR en lugar de DRAW. En esta ocasión se enlazan unas rectas con otras. Al utilizar la línea 1030 para generar A, que puede ser + 1 o − 1, la dirección de cada recta se hace aleatoria. Su longitud es también aleatoria gracias a la línea 1040. El efecto es simular un «camino aleatorio», más o menos el que seguiría una mosca chiflada. La pausa ha sido programada con una instrucción nueva: AFTER 200 GOSUB 1000. Significa que se ejecuta la subrutina cuando el cronómetro ha llegado a 200. Como siempre, para el cronómetro 50 unidades es un segundo, de modo que 200 representa un retardo de 4 segundos. Transcurrido ese tiempo, GOSUB 1000 realiza el dibujo con DRAWR y retorna al bucle infinito de la línea 90.

```
10 MODE 1
20 CLG
30 FOR N=1 TO 50
40 MOVE 320,200
60 DRAW RND(1)*639,RND(1)*399,RND(1)*4
70 NEXT
80 AFTER 200 GOSUB 1000
90 GOTO 90
1000 CLG
1010 MOVE 320,200
1020 FOR N=1 TO 50
1030 A=RND(1):IF A>0.5 THEN A=1 ELSE A=-1
1040 DRAWR A*INT(RND(1)*100),A*INT(RND(1)*100),
     RND(1)*4
1050 NEXT
1060 RETURN
```

Fig. 10.6 Generación de rectas aleatorias para ilustrar las diferencias entre DRAW y DRAWR.

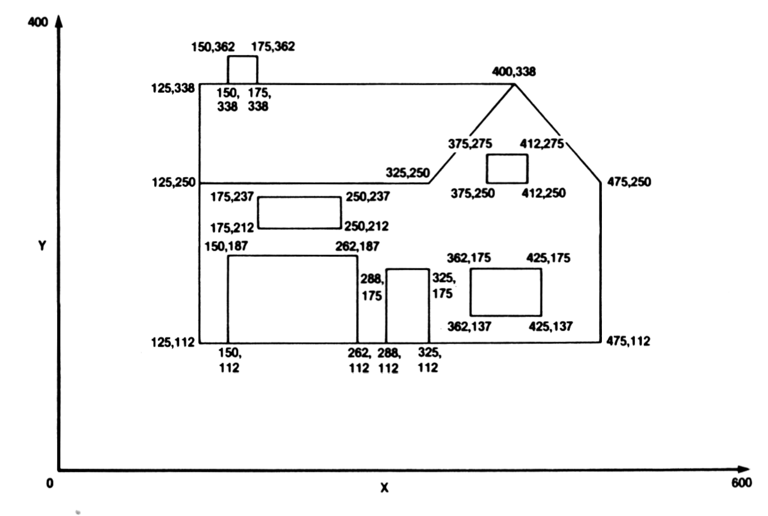


Fig. 10.7 Dibujo de una casa realizado en papel vegetal sobre papel milimetrado.

PLANIFICACIÓN DE LOS GRÁFICOS

Dibujar en gráficos de alta resolución no es difícil; todo lo que se requiere es una buena planificación. El mejor método es utilizar como plantilla una hoja de papel milimetrado. El formato adecuado es A4. Se puede adoptar la escala 4 cm = 100 pi-

xels; entonces la hoja se marca de 0 a 400 en un borde y de 0 a 650 en el otro. Los dibujos se preparan, o bien directamente en la plantilla, o bien en una hoja de papel vegetal superpuesta a ella. Una vez terminado el dibujo (de momento compuesto sólo por rectas), se marcan las coordenadas de los puntos de intersección de las rectas.

```
10 MODE 0
20 MOVE 125,112
30 PEN 2
40 FOR N=1 TO 7
50 READ X,Y
60 DRAW X,Y
70 NEXT
80 MOVE 150,112:FOR N=1 TO 3
90 READ X,Y:DRAW X,Y:NEXT
100 MOVE 288,112
110 FOR N=1 TO 3
120 READ X,Y:DRAW X,Y:NEXT
130 MOVE 362,137
140 FOR N=1 TO 4:READ X,Y
150 DRAW X,Y:NEXT
160 MOVE 375,250
170 FOR N=1 TO 4:READ X,Y
180 DRAW X,Y:NEXT
190 MOVE 175,212
200 FOR N=1 TO 4:READ X,Y
210 DRAW X,Y:NEXT
220 MOVE 150,338
230 FOR N=1 TO 3:READ X,Y
240 DRAW X,Y:NEXT
250 MOVE 125,338:DRAW400,338
1000 DATA 475,112,475,250,400,338,325,250,125,250,
    125,338,125,112
1010 DATA 150,187,262,187,262,112
1020 DATA 288,175,325,175,325,112
1030 DATA 362,175,425,175,425,137,362,137
1040 DATA 375,275,412,275,412,250,375,250
1050 DATA 175,237,250,237,250,212,175,212
1060 DATA 150,362,175,362,175,338
```

Fig. 10.8 Programa que dibuja la casa en la pantalla.

A continuación ya se puede empezar a escribir el programa. Como ejemplo de este método, en la figura 10.7 hemos dibujado una casa. El dibujo se hizo en papel vegetal, con la plantilla debajo, haciendo coincidir las rectas del dibujo con las de la plantilla. Después leímos las coordenadas de los puntos en la plantilla y las anotamos en el dibujo.

El programa resultante es el de la figura 10.8. Hemos procurado servirnos al máximo de las líneas DATA con objeto de que, si el dibujo no resulta convincente, baste con modificar los datos, y no el programa. Cada sucesión de instrucciones DRAW va precedida de una instrucción MOVE. Yo tengo la costumbre de empezar todos los dibujos por su extremo inferior izquierdo. Detrás de cada MOVE viene un grupo de instrucciones DRAW, que son las que en realidad dibujan. Este programa es muy rápido, pero más vale no pensar cuánto se habría tardado en escribirlo sin la adecuada preparación.

DIBUJO DE CIRCUNFERENCIAS

Para muchos tipos de dibujos no bastan las rectas, sino que se requieren circunferencias, elipses y arcos. Aquí nos encontramos con una de las pocas debilidades del CPC464: no tiene una instrucción que dibuje directamente circunferencias, ni tampoco una que rellene recintos con un color determinado. Aceptémoslo como es y utilicemos subrutinas para realizar estas tareas. El manual da muchas sugerencias a este respecto. En la figura 10.9 se muestra un programa que dibuja circunferencias. La línea 10 selecciona el modo de pantalla. La línea 20 pone el origen de coordenadas en el centro de la circunferencia y define el radio, R. La línea 30 invoca la subrutina que dibuja la circunferencia. A continuación se restablece el origen y se entra en el bucle infinito de la línea 50, que se interrumpe pulsando ESC dos veces. En la subrutina, DEG hace que los ángulos se entiendan en grados. En la línea 1010 empieza un bucle que calcula las coordenadas de los puntos de la circunferencia y los dibuja.

Ésta no es la única forma de dibujar una circunferencia, y además es muy lenta,

```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1010 FOR N=1 TO 360
1020 PLOT R*SIN(N),R*COS(N),2
1030 NEXT
1040 RETURN
```

Fig. 10.9 Programa que dibuja una circunferencia. Este método es bastante lento.

```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1005 MOVER 0,R
1010 FOR N=0 TO 360 STEP 10
1020 DRAW R*SIN(N),R*COS(N)
1030 NEXT
1040 RETURN
```

Fig. 10.10 Método mucho más rápido de dibujo de una circunferencia.

aunque fácil de programar. En la figura 10.10 damos una alternativa mucho más rápida. El dibujo no es una verdadera circunferencia, sino una sucesión de rectas que la simulan, esto es, un polígono. El número de lados es bastante elevado (36), por lo que el resultado es razonablemente circular. Hay que elegir entre precisión y velocidad.

Veamos ahora cómo rellenar una circunferencia (es decir, cómo dibujar un círculo). El programa que realiza esta función es el de la figura 10.11, similar al de la figura 10.9. La diferencia está en la subrutina 1000; la idea es trazar una sucesión de rectas, muy próximas unas a otras, cuyos extremos coincidan con los puntos de la circunferencia. Las instrucciones MOVE llevan el cursor a un punto de la izquierda de la circunferencia, y las DRAW trazan rectas hasta el punto correspondiente del lado derecho. Pruebe este método; el resultado es bueno, y la ejecución es casi tan rápida como la de la instrucción FILL de que disponen algunos ordenadores.

```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1010 FOR N=0 TO 360
1020 MOVE -R*SIN(N),R*COS(N)
1030 DRAW R*SIN(N),R*COS(N)
1040 NEXT
1050 RETURN
```

Fig. 10.11 Cómo rellenar una circunferencia. Las figuras rectangulares son mucho más fáciles de rellenar.

TRABAJO DE DETECTIVES

Hay muchos efectos interesantes que se pueden conseguir si se hace que el programa detecte automáticamente la posición del cursor. Recordemos que el cursor gráfico es invisible; su posición se debe determinar por medio de una instrucción. Utilizadas en una línea de programa, las palabras reservadas XPOS e YPOS dan sendos números que son las coordenadas del cursor en el momento en que se ejecuta la instrucción. Equivalen, en este sentido, a POS y VPOS, que daban las coordenadas del cursor en la pantalla de texto. La diferencia es que con XPOS e YPOS no hay que especificar ningún número de ventana. Veamos para qué sirven estas instrucciones.

Una de las cosas que debemos procurar es que el cursor gráfico no se salga de la pantalla. Podemos, por ejemplo, tratar la pantalla gráfica como si fuera una superficie cilíndrica, haciendo que cuando el cursor se salga por un lado aparezca por el otro. La figura 10.12 ilustra este efecto en un programa que dibuja líneas quebradas. La línea 20 define una posición en el centro de la pantalla; la línea 30 coloca el cursor en esa posición. Esta línea contiene una instrucción nueva: RANDOMIZE TIME. Su inclusión es necesaria por lo siguiente: RND no genera verdaderos números aleatorios, sino números pseudoaleatorios tomados de una sucesión que con el tiempo llega a repetirse. Un remedio a este problema consiste en utilizar RANDOMIZE, y mejor aún si después de RANDOMIZE se pone un número que sólo pueda repetirse por azar. TIME es un buen número para este propósito, de modo que RANDOMIZE TIME es un medio casi infalible de evitar que dos ejecuciones distintas de un programa den el mismo resultado.

En la línea 40 empieza un bucle infinito, ya que X nunca puede ser 0. La línea 50 decide si debe dar nuevo valor a X o a Y. Dependiendo del valor de J, se asigna a X o a Y un valor aleatorio. En la línea 60 se utilizan los valores actualizados de X, Y para dibujar una recta con DRAWR. En las líneas 70 a 100 se hace una comprobación de la posición del cursor; si se ha acercado a los bordes de la pantalla,

```
10 CLS
20 X=320:Y=200
30 MOVE X,Y:RANDOMIZE TIME
40 WHILE X<>0
50 J=RND(1):IF J>0.5 THEN X=20*RND(1)ELSE Y=20*
   RND(1)
60 DRAWR X,Y,1
70 IF XPOS>630 THEN MOVE 10,YPOS
80 IF XPOS<10 THEN MOVE 630,YPOS
90 IF YPOS>390 THEN MOVE XPOS,10
100 IF YPOS<10 THEN MOVE XPOS,390
110 WEND
```

Fig. 10.12 Utilización de XPOS e YPOS para asimilar la pantalla a una superficie cilíndrica.

se lo hace saltar al otro lado. Obsérvese cómo hemos utilizado YPOS y XPOS en las instrucciones MOVE. No podíamos poner X,Y, porque estos números son posiciones *relativas*, no absolutas.

Hay otra forma de determinar la posición del cursor gráfico. La función TEST(X,Y) da como resultado un número de INK, que es el correspondiente al punto de coordenadas X,Y. Por lo tanto, se puede incluir TEST(X,Y) en instrucciones IF ... para determinar si el color del punto examinado es el del fondo o el de algún obstáculo. También se puede utilizar TESTR(X,Y), que interpreta sus parámetros como coordenadas relativas. Por ejemplo, en modo 2 TESTR(1,0) examina el pixel que está inmediatamente a la derecha del cursor gráfico. En modo 1 habría que poner TESTR(2,0); en modo 0, TESTR(4,0).

CONEXIÓN DEL CURSOR DE TEXTO CON EL CURSOR GRÁFICO

Nuestra pantalla gráfica ha sido hasta ahora demasiado estática, limitada al dibujo de curvas y rectas sin movimiento. Para animarla vamos a utilizar la instrucción TAG. El efecto de esta instrucción es conectar el cursor de texto al cursor gráfico. Pero el cursor gráfico es muy pequeño, un pixel, mientras que el de texto es una matriz de 8 × 8 pixels. El punto de contacto de los dos cursores es el extremo superior izquierdo del cursor de texto. Si enviamos a la pantalla gráfica un carácter con una instrucción PRINT, éste se extenderá 8 pixels a la derecha y 8 pixels hacia abajo con respecto a la posición del cursor gráfico. La virtud de TAG es que nos permite escribir con CHR$ caracteres cualesquiera (incluso los definidos por nosotros) *en la posición del cursor gráfico*. ¿Y para qué sirve esto?, se preguntará el lector. Pues bien, si tenemos en cuenta que en una línea de texto en modo 1 sólo disponemos de 40 posiciones posibles para colocar un carácter, los saltos que éste tiene que dar cuando lo movemos son tan grandes que el movimiento resulta demasiado torpe. En cambio, el cursor gráfico se mueve a saltos mucho más pequeños. En modo 1 recorre la pantalla en 320 pasos, utilizando valores de X comprendidos entre 0 y 639. Este movimiento es ya suficientemente fino como para hacer la animación convincente.

Veamos como ejemplo el programa de la figura 10.13. El TAG de la línea 30 conecta los dos cursores, lo que nos permite escribir caracteres en la posición del cursor. En la línea 40 empieza un bucle que mueve el cursor gráfico a través de la pantalla. La línea 50 sitúa el cursor en la posición correspondiente a X,Y. Obsérvese que hemos puesto STEP 2 en el bucle. Ello se debe a que en modo 1 sólo hay 320 posiciones horizontales distintas, a pesar de que los valores de X varían de 0 a 639. Cada posición del cursor se corresponde con dos números: X = 0 y X = 1 son una misma posición; X = 2 y X = 3 son la posición siguiente por la derecha, etc. En cambio, en modo 2 cada valor de X representa una posición; pruebe el programa en este modo. En modo 0 sólo hay 160 posiciones distintas por línea, de forma que los valores 0, 1, 2 y 3 de X representan la misma posición, y la siguiente es X = 4, 5, 6 o 7, etc. En modo 0 tendríamos que poner STEP 4 en el bucle. Estos números de STEP eliminan otra causa de brusquedad en el movimiento.

En la posición del cursor escribimos un espacio en blanco. La línea 70 mueve el

```
10 MODE 1
20 Y=200
30 TAG
40 FOR X=0 TO 639 STEP 2
50 MOVE X,Y
60 PRINT" ";
70 MOVER 2,0
80 PRINT CHR$(243);
90 CALL &BD19
100 NEXT
110 TAGOFF
120 END
```

Fig. 10.13 La instrucción TAG permite escribir caracteres en la posición del cursor gráfico.

cursor a la derecha, y la línea 80 escribe el carácter 243. Aquí el punto y coma es imprescindible. Ejecute el programa sin él y verá lo que ocurre. Al escribir en la pantalla gráfica, se escribe todo, incluso los códigos de retorno al principio de la línea y de avance a la línea siguiente. Cuando se utiliza TAG, estos códigos se suprimen mediante el punto y coma.

El siguiente misterio está en la línea 90. Esta instrucción tiene que ver con el «código de máquina». La mejor forma de apreciar su efecto es suprimir la línea y ejecutar el programa. El carácter se moverá más deprisa, pero con ciertos efectos peculiares que aparecen de vez en cuando. Estas distorsiones se producen porque la imagen de la pantalla se forma mediante un punto que se mueve describiendo líneas horizontales que evolucionan de arriba abajo, y que se ilumina cuando deba aparecer algo en la pantalla. Cuando el carácter se está moviendo por la pantalla, los dos movimientos entran en conflicto. Para evitar este problema hay que sincronizar la velocidad de movimiento del carácter con la velocidad a la que el punto forma la imagen (50 veces por segundo). De ello se encarga una rutina de la ROM del CPC464. La instrucción que la invoca es CALL &BD19. "&BD19" es un número escrito en un sistema de numeración denominado *hexadecimal*. Si esto despierta su curiosidad, espere mi próximo libro, *Introducción al código de máquina para el Amstrad*.

El lector pensará que limitar la velocidad del movimiento es precisamente lo que no queremos hacer. ¿Cómo compensar esta pérdida de velocidad? Una forma es hacer que X e Y sean enteros, X % e Y %. Si esto es suficiente para su gusto, pruebe con STEP 4, o incluso con STEP 8. Pero no omita CALL &BD19: la falta de sincronización es mucho más evidente cuando el carácter se está moviendo deprisa. Observe que en este programa la línea 70 sirve para no borrar completamente el carácter, lo que también ayuda a reducir la brusquedad del movimiento. Si el diseño del carácter fuera distinto, se podría suprimir esta instrucción, o quizá utilizar un desplazamiento distinto.

ANIMACIÓN DE LAS TINTAS

Hay otro método para mover objetos por la pantalla o, mejor dicho, para simular su movimiento. Se basa en el manejo de la instrucción INK. Supongamos que lo que queremos mover no es un carácter, sino un dibujo completo creado con MOVE y DRAW. Para moverlo se lo puede dibujar, borrarlo y dibujarlo de nuevo en otra posición, y así sucesivamente. Pero entonces el movimiento será lento y brusco. El método alternativo consiste en dibujar la forma deseada en todas las posiciones que deba ocupar al moverse, asignándole en cada posición tinta de un tintero diferente. Inicialmente todos los tinteros contienen la misma tinta que el fondo, de manera que el dibujo es invisible. A continuación se cambia la tinta de un tintero a un color distinto del existente en el fondo. Esto hará visible uno de los dibujos. Se introduce un retardo y se vuelve a cambiar la tinta al color del fondo. El dibujo desaparece, pero entonces se cambia la tinta del siguiente tintero, con lo que otro dibujo se hace visible, y así sucesivamente. El efecto es de un movimiento cuya velocidad se puede regular fácilmente. El modo adecuado para este método es el 0, pues en él se dispone de más tinteros que en los otros modos.

```
10 MODE 0
20 FOR CL=2 TO 15
30 INK CL,1:NEXT
40 FOR NR=1 TO 4
50 READ X,Y:MOVE X,Y
60 FOR LN=1 TO 4
70 READ X,Y
80 DRAW X,Y,NR+1
90 NEXT:NEXT
100 WHILE INKEY(47)=-1
110 FOR NR=2 TO 5
115 CALL &BD19
120 INK NR,24
130 FOR X=1 TO 30:NEXT
140 INK NR,1
150 NEXT
160 WEND
170 DATA 300,300,350,300,350,100,300,100,300,300
180 DATA 385,288,415,252,270,113,235,148,385,288
190 DATA 225,225,425,225,425,175,225,175,225,225
200 DATA 230,255,275,293,410,148,377,110,230,255
210 MODE 1:END
```

Fig. 10.14 Animación con tintas invisibles.

La figura 10.14 da un ejemplo de este artificio. La primera parte del programa pone en los tinteros 2 a 15 la misma tinta que en el fondo, que es de color 1. Las líneas 40 a 90 dibujan cuatro rectángulos; uno es vertical, y los otros están girados cada uno 45 grados con respecto al anterior. Éstas son las sucesivas posiciones de un rectángulo que gira en el plano en el sentido de las agujas del reloj. Si el lector quiere observar cómo se dibujan los triángulos, ponga en la línea 30 el número 23 para el color de la tinta e introduzca la línea 95 STOP.

Cuando se termina de dibujar los rectángulos, con la pantalla aún vacía, se los puede visualizar cambiando los colores de las tintas. El bucle que empieza en la línea 100 continúa hasta que se pulsa la barra espaciadora. A partir de ahí el bucle FOR... NEXT va cambiando los colores sucesivamente y devolviéndolos al color del fondo.

Nada se consigue sin esfuerzo, y menos cuando se trabaja con gráficos. En este ejemplo había que calcular las coordenadas de los vértices de los rectángulos y ponerlas en las líneas de DATA; éste es un trabajo muy pesado, que se simplifica si se dispone de un tablero de dibujo. No obstante, es muy probable que alguien escriba y comercialice un programa que realice estos cálculos. Es un trabajo de geometría que la máquina puede realizar mucho más deprisa que usted.

11

Los sonidos del CPC464

La capacidad de producir sonidos es una característica esencial de todos los ordenadores modernos. El sonido del CPC464 proviene del altavoz que tiene incorporado; el control del volumen está en la cara derecha de la carcasa del ordenador. En la cara posterior de ésta hay un zócalo a través del cual se puede conectar el generador de sonido a un amplificador de alta fidelidad, lo que permite escuchar los sonidos a mayor volumen y con mejores graves. La diferencia es asombrosa. Además, el sonido puede ser estereofónico si se utiliza un amplificador externo. También se pueden conectar unos auriculares como los que se utilizan con los reproductores de cassette portátiles.

```
10 CLS:PRINT TAB(19)"MENU"
20 PRINT:PRINT TAB(3)"1. MUSICA"
30 PRINT TAB(3)"2. EXPLOSIONES"
40 PRINT TAB(3)"3. RUIDOS ACUATICOS"
50 PRINT TAB(3)"4. MOTORES"
60 PRINT:PRINT"ELIJA UN NUMERO"
70 GOSUB 1000
80 IF J<1 OR J>4 THEN PRINT"INCORRECTO. ELIJA OTRA
   VEZ.":GOTO 10
90 ON J GOSUB 500,500,500,500
100 END
500 PRINT"AQUI SE PONDRA UN PROGRAMA MAS TARDE"
510 RETURN
1000 K$=INKEY$:IF K$<>""THEN J=VAL(K$):RETURN
1010 PRINT CHR$(7);:FOR N=1 TO 500:NEXT
1020 GOTO 1000
```

Fig. 11.1 Producción de un pitido con CHR$(7).

Si todo lo que se necesita en un programa es un pequeño pitido de aviso, por ejemplo, en un menú, basta con programar PRINT CHR$(7); este carácter produce el pitido. En la figura 11.1 se da un ejemplo. Este listado es el principio de un programa (imaginario) que demuestra diversos tipos de sonidos. La zona del programa que nos interesa ahora es la subrutina de la línea 1000. La instrucción K$ = INKEY$ detecta la pulsación de una tecla; cuando ésta se produce, a J se asigna el valor VAL(K$). Veamos qué ocurre cuando no se ha pulsado ninguna tecla: la rutina continúa en la línea 1010, donde CHR$(7) produce un pitido. Esta instrucción tiene que terminar con un punto y coma para evitar el desplazamiento de la pantalla; aunque PRINT CHR$(7) no escribe nada en la pantalla, toda instrucción PRINT que no termine con punto y coma provoca el salto del cursor a la línea siguiente. Tras un pequeño retardo, la instrucción GOTO 1000 inicia una nueva comprobación con INKEY$. El resultado global es que la máquina continúa pitando hasta que se pulse una tecla; ésta es una alternativa al método del asterisco parpadeante, con el que se puede combinar para asegurar que el usuario no ignore el menú.

Si el lector quiere ir un poco más lejos en este terreno, será necesario que conozca algunos hechos acerca de la física de los sonidos. El fenómeno que denominamos *sonido* es el resultado de las variaciones rápidas de presión en el aire cercano a nues-

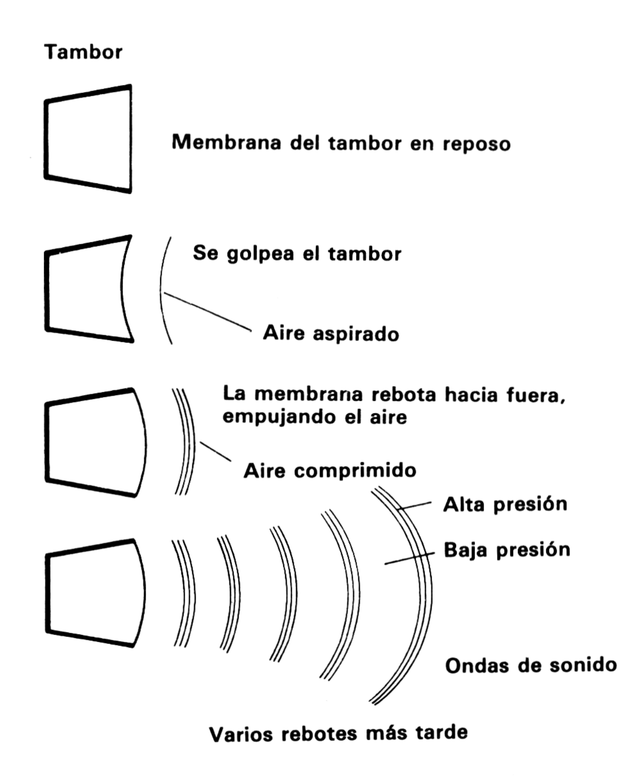


Fig. 11.2 Generación de sonido por un tambor. La membrana produce compresión y descompresión del aire alternadamente.

tros oídos. Todos los sonidos se producen por generación de esas variaciones de presión. En la figura 11.2 se ilustra cómo produce su sonido un tambor. Las variaciones de presión del aire son invisibles; el oído tampoco las detecta si no son suficientemente rápidas. La frecuencia de las variaciones se mide en *hertzios* (ciclos por segundo). Un ciclo en una onda es el tiempo que ésta tarda en aumentar la presión hasta el máximo, volver a la presión normal, disminuir la presión hasta el mínimo y volver por segunda vez a la presión normal (Fig. 11.3). La razón por la que utilizamos el término *onda* es la forma que tiene la gráfica de la presión en función del tiempo, similar a la de las ondas en la superficie del agua.

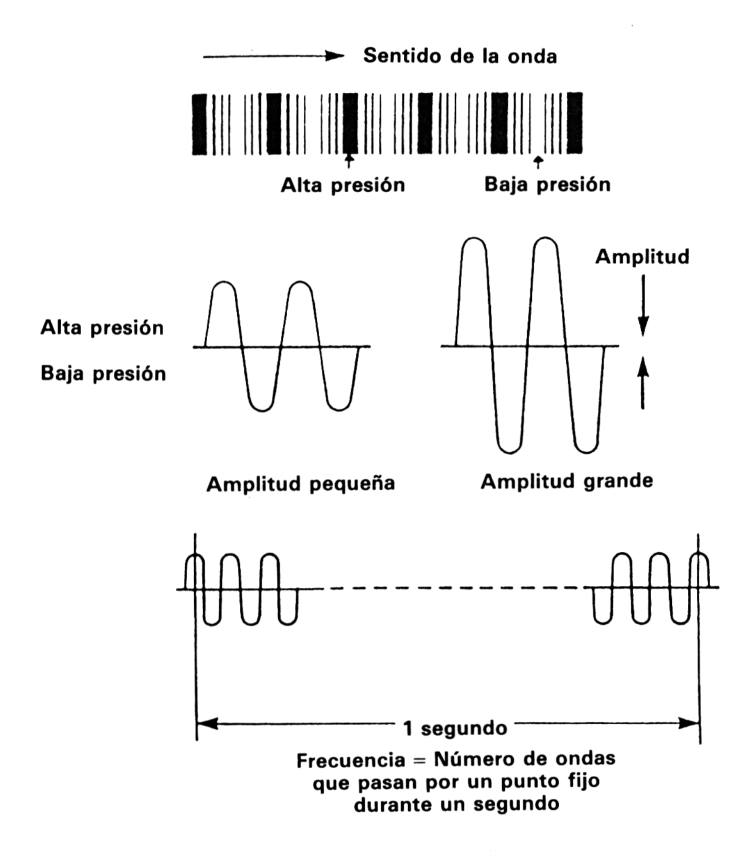


Fig. 11.3 Frecuencia y amplitud de una onda de sonido.

La frecuencia de un sonido se mide en hertzios (Hz), número de ciclos de variación de la presión por segundo. Si la frecuencia es inferior a 20 Hz, el sonido es inaudible, aunque puede tener otros efectos. El margen de frecuencias audibles se extiende desde los 20 hasta más de 15 000 Hz. La frecuencia de las ondas es lo que percibimos como «tono» de una nota. Las frecuencias de 80 a 120 Hz corresponden a lo

que entendemos por tonos graves; los tonos agudos son los que tienen frecuencias superiores a los 400 Hz.

La amplitud de las variaciones de presión determina la *intensidad* de una nota. La *amplitud* de una onda es la máxima variación de la presión con respecto al valor normal. Para controlar plenamente la generación del sonido, tenemos que especificar la amplitud, la frecuencia y la forma de la onda, así como la forma de variación de la amplitud durante el tiempo que la nota esté sonando.

Al escribir música, el compositor tiene que especificar la intensidad, la duración y el tono de cada nota. En la música escrita, la intensidad de las notas se indica con las letras *f* (forte, intenso) y *p* (piano, suave). Por ejempo, *fff* significa fortísimo (muy intenso) y *ppp* significa pianísimo (muy suave). La duración de las notas se indica de dos formas distintas. Una de ellas se basa en el funcionamiento del metrónomo. Este aparato es un generador de sonido que produce un «tic» a intervalos regulares; del ajuste del metrónomo depende el número de batidos que realiza por minuto. La unidad de duración de las notas es la nota *negra*, de forma que el ajuste del metrónomo decide cuántas negras deben sonar por minuto. La duración de una nota se expresa por comparación con esta unidad. Así, una *mínima* o *blanca* dura el doble que una negra, y una *semibreve* o *redonda* dura el doble que una mínima. Una *corchea* es la mitad de una negra; una *semicorchea*, la cuarta parte. En la figura 11.4 se muestran las posibles duraciones de las notas y los signos correspondientes. Hay otros signos que se emplean para representar la duración de los silencios (Fig. 11.5). El otro método de especificar la duración de las notas es bastante menos preciso; utiliza palabras italianas tales como lento, moderato y allegro.

Símbolo	Duración	Nombre
𝅘𝅥𝅰	1/8	Fusa
𝅘𝅥𝅯	1/4	Semicorchea
𝅘𝅥𝅮	1/2	Corchea
𝅘𝅥	1	Negra
𝅗𝅥	2	Blanca
o	4	Redonda

Fig. 11.4 Símbolos que se utilizan en las partituras para indicar la duración de las notas.

El tono de una nota se indica por la altura a que se la escribe en el *pentagrama*. La música para piano se escribe en dos pentagramas paralelos, cada uno de los cuales está formado por cinco rayas y cuatro espacios. El pentagrama superior es el de agudos (clave de sol) y el inferior, el de graves (clave de fa). La música para instrumentos que no tienen teclado se escribe en un solo pentagrama. Para el piano y el órgano se utilizan, como hemos dicho, dos pentagramas. Entre ellos queda el espacio para la nota *do media*. En el piano esta nota se obtiene de la tecla que está en el centro del teclado. En la figura 11.6 se muestran los dos pentagramas.

Símbolo del silencio	Duración
𝄿	¼
𝄾	½
𝄽	1
𝄼	2
𝄻	4

Fig. 11.5 Símbolos de los silencios en las partituras.

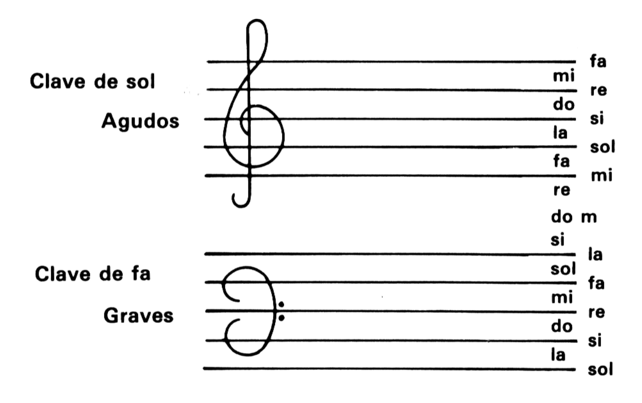


Fig. 11.6 Los dos pentagramas, con los nombres de las notas. Obsérvese la posición de la nota do media.

Las notas que se muestran en la figura 11.6 se disponen en grupos de 8. Cada grupo forma una *octava*. También hay notas de tono intermedio entre cada dos de estas notas: son los *semitonos*. En la música occidental se divide la octava en un total de doce notas, entre tonos y semitonos. En la figura 11.7 se muestra su colocación en el teclado del piano. Los semitonos son los correspondientes a las teclas negras. En las partituras los semitonos se indican mediante los signos # (sostenido) y ♭ bemol. Un sostenido indica un semitono por encima del tono de la nota; un bemol, un semitono por debajo. En el piano, un semitono por encima de una nota es igual a un bemol por debajo de la siguiente, de modo que do sostenido es lo mismo que re bemol. Esto no es necesariamente cierto en otros instrumentos.

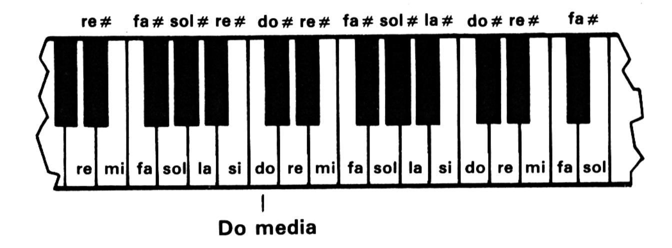


Fig. 11.7 Parte del teclado del piano, en la que se indica la posición del do medio. Sólo hay un semitono entre si y do y entre mi y fa.

INSTRUCCIONES DE SONIDO DEL CPC464

El CPC464 dispone de una instrucción de sonido que puede utilizar de distintas formas, desde la más sencilla a la más compleja. Todo depende del efecto que se desee conseguir. No se requiere gran planificación si lo que se pretende es producir una nota de aviso o ejecutar una sencilla tonada. A medida que se va adquiriendo experiencia, se le van encontrando aplicaciones más interesantes. De todos los ordenadores que conozco, el CPC464 es con gran diferencia el que tiene el sistema de sonido más potente, desde el punto de vista de poder controlar los sonidos con instrucciones de BASIC. Empecemos con la simple generación de notas.

La instrucción específica es SOUND. Esta instrucción debe ir seguida de al menos dos parámetros. El primero es un número que selecciona los canales. El CPC464 puede producir tres notas al mismo tiempo, y las tres pueden ser controladas por separado asignándolas a tres canales distintos. Los canales son A, B y C y pueden ser seleccionados uno por uno o agrupados de todas las formas posibles. La figura 11.8 da la relación de los números de selección junto con los canales que seleccionan. Uno de los aspectos en los que el CPC464 supera a los demás ordenadores es la posibilidad de generar sonido estereofónico. Cuando se amplifican las señales con un amplificador externo, el canal A es enviado al altavoz izquierdo, el C al derecho

Número	Canal seleccionado
1	A
2	B
3	A y B
4	C
5	A y C
6	B y C
7	A y B y C

Fig. 11.8 Números que se utilizan en la instrucción SOUND para seleccionar canales.

y el B a ambos. Los números de selección superiores al 7 producen efectos complejos que examinaremos más adelante.

El segundo parámetro de la instrucción SOUND es el «número de tono». Este número controla el tono de la nota generada por el CPC464; su margen va de 1 (nota más aguda) a 4095 (nota más grave). El margen utilizable en la práctica es 10-1000, ya que las notas ajenas a este margen no producen más que ruidos, salvo que se las reproduzca en un buen sistema de alta fidelidad. Se puede utilizar el número 0, pero con otro propósito que más tarde explicaremos. Las frecuencias equivalentes a estos números están especificadas en el manual, apéndice VII. La tabla de la página dos de ese apéndice cubre las notas más utilizadas: la escala de do mayor, desde do medio hasta una octava por encima de esta nota. El teclado del piano es muy adecuado para reproducir estas notas. La escala empieza en do medio y termina en otra nota do, pero en la octava siguiente.

Comencemos nuestra investigación de la instrucción SOUND con una nota sencilla (Fig. 11.9). La primera instrucción SOUND aparece en la línea 10; el canal seleccionado es el A, gracias al parámetro 1; el número de tono es el 478, que corresponde a una nota muy próxima a do medio. La línea 10 pone en marcha el sonido; el efecto de esta instrucción es hacer sonar la nota durante un cincuentavo de segundo.

```
10 SOUND 1,478,50
20 FOR N=1 TO 2000:NEXT
30 SOUND 1,478,100
40 FOR N=1 TO 2000:NEXT
50 SOUND 1,478,200
```

Fig. 11.9 Generación de notas aisladas.

Ahora es *muy importante* comprender lo que está sucediendo. Cuando el ordenador ejecuta una instrucción SOUND, toma de ella los parámetros y los envía a un sistema autónomo, el sistema de sonido. Esta parte del ordenador funciona por sí misma y genera el sonido. Mientras tanto, el programa continúa a partir de la instrucción siguiente. Si hay muchas instrucciones SOUND, el ordenador las recorre a gran velocidad y envía inmediatamente los parámetros al sistema de sonido, para seguir luego ejecutando el resto del programa. Cada nota tarda cierto tiempo en ser ejecutada; el ordenador puede haber enviado los datos de cinco notas más antes de que la primera haya dejado de sonar. Los datos habrán sido enviados a la «cola de sonidos»; si no se comprende esta idea, se encontrará que las instrucciones SOUND más complejas no producen los efectos de ellas esperados. En este ejemplo se ha introducido un retardo después de cada instrucción SOUND. Los retardos son suficientemente largos como para que las notas hayan terminado de sonar antes de que el ordenador haya podido ejecutar la siguiente instrucción SOUND. La duración de cada nota la especifica el tercer parámetro de la instrucción. En la línea 30, este parámetro es el número 50, lo que equivale a 50/100 segundos, es decir, medio segundo. Los valores cero y negativos tienen significados especiales que veremos más adelante.

```
10 SOUND 1,478
20 WHILE SQ(1)>128:WEND
30 FOR N=1 TO 100:NEXT
40 SOUND 1,478,50
50 WHILE SQ(1)>128:WEND
60 FOR N=1 TO 100:NEXT
70 SOUND 1,478,200
```

Fig. 11.10 Generación en notas con una pausa fija entre ellas. SQ determina cuándo una nota ha dejado de sonar.

La figura 11.10 muestra una modificación del programa anterior. La instrucción nueva es SQ, que informa sobre la situación de la cola de sonidos de la misma forma que XPOS lo hace sobre la posición del cursor. Mientras haya en la cola algún sonido en espera ser ejecutado, SQ dará un valor igual o superior a 128. Comprobando este valor en un bucle WHILE... WEND podemos hacer que el ordenador espere a que se vacíe la cola, de modo que no envíe a ella datos nuevos mientras esté sonando la nota anterior. Ésta es otra forma de separar unas notas de otras. El bucle FOR... NEXT da siempre el mismo intervalo entre el principio de dos notas sucesivas, independientemente de la duración de éstas; en cambio, al utilizar SQ en un bule WHILE... WEND se inicia cada nota al final de la anterior. SQ tiene como parámetro el número de selección de canal entre paréntesis. Las otras aplicaciones de SQ, como ocurre con muchas de estas instrucciones de sonido, requerirían un manual entero para sí solas.

```
10 CLS:PRINT TAB(13)"LA ESCALA DE DO MAYOR":PRINT:
   PRINT
20 FOR N=1 TO 8
30 READ NOTA
40 SOUND 2,NOTA,100
50 NEXT
60 PRINT"AHORA TODAS LAS NOTAS ESTAN EN LA COLA"
100 DATA 478,426,379,358,319,284,253,239
```

Fig. 11.11 La escala de do mayor, intepretada por el CPC464.

El siguiente paso va a consistir en generar una escala musical. La tabla de números y frecuencias del apéndice del manual indica qué números de tono debemos emplear. Consultando esa tabla hemos escrito el programa de la figura 11.11, que interpreta la escala de do mayor. Esta escala empieza en do medio y termina en el do de la octava siguiente. Obsérvese que el paso de una nota cualquiera de una octava

a la misma nota de la octava siguiente representa duplicar la frecuencia. Dividir por dos la frecuencia es bajar una octava. Los números de tono del CPC464 operan de forma inversa. Como se puede apreciar en la figura 11.11, el do medio corresponde al número 478, y el do de la octava siguiente corresponde al número 239. Dado que estos números son enteros, no siempre se obtienen las frecuencias exactas al dividirlos por dos.

La figura 11.11 también ilustra el funcionamiento de la cola de sonidos. El CPC464 puede almacenar hasta cinco notas en cada cola. Una de ellas estará sonando, y las otras cuatro esperando. Al ejecutar este programa se verá cómo aparece el título en la pantalla y empieza a sonar la primera nota. Cuando se han enviado cinco notas a la cola y hay otras por leer, el ordenador también tiene que pararse a esperar. En cuanto haya podido enviarlas todas, continuará con el resto del programa y escribirá el siguiente mensaje.

Veamos ahora cómo se pueden utilizar varios canales al mismo tiempo. El programa de la figura 11.12 ejecuta un acorde de tres canales. Todavía no hemos hablado

```
10 SOUND 1,478,300
20 SOUND 2,379,300
30 SOUND 4,239,300
```

Fig. 11.12 Acorde con tres canales.

de cómo controlar el volumen; por simplificar hemos hecho las tres notas de igual intensidad, pero los mejores efectos musicales se consiguen dando a cada nota la intensidad adecuada. Dado que el oído humano es menos sensible a las frecuencias bajas y altas que a las medias, frecuentemente se hace que las frecuencias extremas suenen con mayor intesidad. El acorde no resulta tan armonioso para el oído educado como debería ser, ya que los números que determinan los tonos no dan las frecuencias musicalmente más agradables. Por ejemplo, para mi gusto el número 380 (línea 20) es preferible al 379.

El cuarto parámetro de la instrucción SOUND es el «número de intensidad». Este número especifica la intensidad relativa de las notas, en el sentido de que si a cuatro notas se les asignan números de intensidad diferentes, éstas sonarán con intensidades diferentes para una misma posición del potenciómetro de volumen del CPC464 (o del amplificador externo). El valor absoluto se controla, pues, con ese potenciómetro. El margen de valores son los enteros del 0 (silencio) al 7 (máxima intensidad). Se pueden utilizar los números del 0 al 15 cuando la intensidad está siendo controlada por una *envolvente*, concepto éste que explicaremos más adelante. Si se intenta utilizar números mayores que el 15, el ordenador emite el mensaje «Improper argument».

En la figura 11.13 se da un ejemplo de utilización de los números de intensidad. Las tres primeras instrucciones forman un bucle en el que se va incrementando el volumen de una nota. En el otro bucle se consigue el efecto inverso. Las notas producidas por los elementos musicales experimentan variaciones de intensidad de este

```
10 FOR N=0 TO 7
20 SOUND 2,339,50,N
30 NEXT
40 FOR N=7 TO 0 STEP -1
50 SOUND 2,339,50,N
60 NEXT
```

Fig. 11.13 Control de la intensidad de las notas.

tipo. Muchos instrumentos producen notas que empiezan con intensidad considerable y luego se van atenuando. El piano es uno de ellos, pues la intensidad máxima se produce cuando se golpea la cuerda, y luego se va atenuando por efecto de la presión del «apagador». Pero el CPC464 permite controlar las variaciones de intensidad (es decir, la *envolvente*) sin necesidad de recurrir a los bucles.

MÚSICA

La capacidad del CPC464 de producir sonido en varios canales permite componer la música con armonía. Si el lector no es compositor de música, lo mejor que puede hacer es tomar como guía una partitura. La música preferible es la escrita para violín y piano o flauta y piano. Estos instrumentos operan en unos márgenes de frecuencia que el pequeño altavoz del ordenador puede manejar relativamente bien. Es evidente que la música para contrabajo no puede sonar bien en un altavoz tan pequeño. Evite la música escrita para instrumentos tales como el clarinete y el bajón, en los que las notas suenan de forma distinta de como lo harían en otro instrumento que ejecutara la misma música.

La mejor técnica de programación consiste en construir un bucle que lea en líneas DATA los números de tono y las duraciones de las notas. Más adelante nos ocuparemos de los otros datos que se pueden incluir en la instrucción SOUND. Cuando se empieza a practicar con música, es conveniente poner los datos de las diversas notas en líneas de DATA distintas. Si se desea que una nota continúe sonando en un canal mientras se modifican las notas de los otros, es preciso utilizar los métodos de sincronización que estudiaremos más adelante. La música de piano suena mejor si se introduce cierta separación entre notas; pero el órgano no necesita tal cosa. La experiencia le permitirá refinar los programas una vez aprendidas las técnicas básicas.

Veamos por ejemplo el programa de la figura 11.14, que ilustra la armonía para tres voces. Este programa no se escribió siguiendo una partitura, sino probando los canales uno a uno. El programa es bastante sencillo; no es más que un bucle en el que se leen los datos necesarios. Las variables C1, C2 y C3 representan los tonos de las notas que se envían a los canales A, B y C. La variable P especifica la duración. Si se programa de esta forma es fácil modificar la música alterando las líneas de DATA. Yo empecé programando un canal; cuando conseguí la melodía, añadí el segundo canal, y después el tercero. El resultado no es tan perfecto como si hubie-

```
10 FOR N=1 TO 9
20 READ C1,C2,C3,P
30 SOUND 1,C1,P,7
40 SOUND 2,C2,P,5
50 SOUND 4,C3,P,5
60 NEXT
70 END
500 DATA 478,319,956,100
510 DATA 506,319,851,70
520 DATA 478,319,956,210
530 DATA 426,319,851,70
540 DATA 379,253,758,140
550 DATA 319,268,638,70
560 DATA 358,284,716,70
570 DATA 379,319,758,100
580 DATA 426,319,851,140
```

Fig. 11.14 Un poco de armonía, en tres canales.

ra utilizado una partitura. Después de todo, Beethoven escribió la música, yo no hice más que programar la máquina.

EFECTOS ESPECIALES

La instrucción SOUND puede producir, aun utilizándola en esta forma tan sencilla, gran variedad de efectos sonoros. Empecemos con una serie de notas de frecuencia creciente. Este efecto puede servir como llamada de atención, o aviso de que «algo está a punto de suceder». Consideremos el ejemplo de la figura 11.15. El bucle externo que empieza en la línea 10 hará que se repita la nota diez veces. En el bucle interno, el número de tono N varía de 200 hasta 0 en saltos de −10. De esta forma se cubre buena parte del espectro audible, con una variación muy rápida del tono. La

```
10 FOR J=1 TO 10
20 FOR N=200 TO 1 STEP -10
30 SOUND 2,N,1,7
40 NEXT
50 NEXT
```

Fig. 11.15 Programación de una nota de tono creciente.

duración de las notas individuales es muy pequeña, y el volumen, el máximo. Ejecute el programa y escuche con atención. Uno de los problemas de las instrucciones de sonido es que resulta muy difícil describir con palabras su efecto. A diferencia de los programas gráficos, en los que uno se puede hacer buena idea de qué va a ocurrir con sólo echar un vistazo al listado, los programas de sonido casi siempre dan una sorpresa, a menos que se sea muy experto en ellos. Recomendamos al lector que compruebe todos estos pequeños programas con objeto de que vaya adquiriendo experiencia. Una buena idea es guardar copias de los sonidos, para luego poder reproducirlos en un cassette corriente.

```
10 FOR J=1 TO 50
20 SOUND 2,239,5,7
30 SOUND 2,244,5,7
40 NEXT
```

Fig. 11.16 Programación de un trino.

El programa de la figura 11.16 produce un trino. Un sonido de este tipo es útil para llamar la atención del usuario, o para anunciar la inminencia de un suceso en un juego. Por alguna razón, las notas de este tipo atraen mejor la atención que las notas sencillas; por eso se las utiliza en los teléfonos modernos. El programa empieza con bucle de la línea 10. Se generan 50 pares de notas, de una duración 5/100 segundos. Los números de tono elegidos para este ejemplo son 239 y 244. Cuanto más agudo sea el tono, más eficaz será el sonido en cuanto a llamar la atención. Pruebe con los números 90 y 95.

RUIDO

Hasta ahora hemos programado solamente la generación de notas musicales ordinarias. Pero también podemos mezclarlas con ruido. *Ruido* es una mezcla aleatoria de frecuencias, si bien en la práctica se observa que hay una frecuencia (o margen de frecuencias) que predomina sobre las demás. Disponer de un generador de ruido es interesante, pues nos permite generar efectos sonoros tales como oleaje, tambores y disparos, que son imposibles de obtener con la instrucción SOUND ordinaria. Para producir ruido tenemos que cerrar el generador de tonos poniendo como número de tono el 0. El «número de tono de ruido» puede variar entre 0 y 15, y tiene que ser el séptimo parámetro de la instrucción SOUND. Nos hemos saltado los parámetros quinto y sexto, pero los explicaremos más adelante. Por el momento, los haremos iguales a cero.

El programa de la figura 11.17 envía ruido al canal B. Como sólo hay un generador de ruido, no se puede enviar ruidos diferentes a los tres canales, aunque sí se puede elegir en qué canal debe sonar el ruido. Con un amplificador estereofónico y un par de altavoces, se puede hacer una visita a Dodge City, con disparos por la

```
10 CLS
20 FOR J=0 TO 15
30 PRINT"PERIODO DEL RUIDO";J
40 SOUND 1,0,20,7,0,0,J
50 FOR N=1 TO 2000:NEXT
60 NEXT
```

Fig. 11.17 Generación de ruido en un canal.

derecha y por la izquierda. El número que especifica el «periodo de ruido» no tiene ni el mismo margen ni el mismo significado que en los otros canales. El margen va de 0 a 15. Todos estos números producen efectos distintos, pero no hay mucha diferencia entre los producidos por los números 0 a 5. Pruebe el programa de la figura 11.18. Produce dos efectos espléndidos mediante la generación de ruido en dos bucles, en cada uno de los cuales se varía un parámetro diferente. En el primero se

```
10 CLS:PRINT:PRINT"OLEAJE EN LA PLAYA..."
20 FOR N=1 TO 10
30 FOR J=15 TO 1 STEP -1
40 SOUND 2,0,20,7,0,0,J
50 NEXT:NEXT
60 PRINT:PRINT"EL HOJALATERO..."
70 FOR N=1 TO 10
80 FOR J=15 TO 0 STEP -1
90 SOUND 2,0,5,J,0,0,1
100 NEXT:NEXT
```

Fig. 11.18 Producción de efectos sonoros con el generador de ruido.

producen diez ruidos de «oleaje» variando el número de periodo de ruido (variable J). Al principio el predominio es de las frecuencias bajas y después se va desplazando hacia las altas. La segunda parte del programa muestra cómo se pueden imitar martillazos sobre metal. En este caso el bucle modifica la intensidad, pero el periodo de ruido permanece constante. Al ser pequeño este valor, el efecto es de martillazos sobre hojalata. Poniendo otros números mayores se obtienen efectos diferentes.

FORMAS DE ONDA

Ha llegado el momento de averiguar cómo se controla con la instrucción SOUND la envolvente de los sonidos. El efecto no es nada sencillo, y por consiguiente es difí-

cil de describir con palabras; así pues, ejecute los programas y escuche con atención. Antes de nada, expliquemos qué es una *envolvente*. Los sonidos que produce la instrucción SOUND, tal como la conocemos hasta ahora, tienen amplitud y frecuencia constantes. Es decir, su intensidad y su tono no varían durante la ejecución de la nota. Pero las notas producidas por los instrumentos musicales no son así. Por ejemplo, una nota de piano es más intensa al principio, inmediatamente después de que el martillo golpee la cuerda, y luego se atenúa muy deprisa. Esto es lo que hace tan inconfundibles los sonidos del piano. Otros instrumentos producen las notas de forma muy diferente, y todos los instrumentos producen notas que no son tonos puros, sino mezclas de diversas frecuencias. Estas diferencias son las que permiten distinguir un piano de un violín y éste de una flauta, aunque todos ellos estén interpretando la misma nota. Una gráfica de la amplitud de una nota en función del tiempo es lo que se denomina «envolvente de volumen» de la nota.

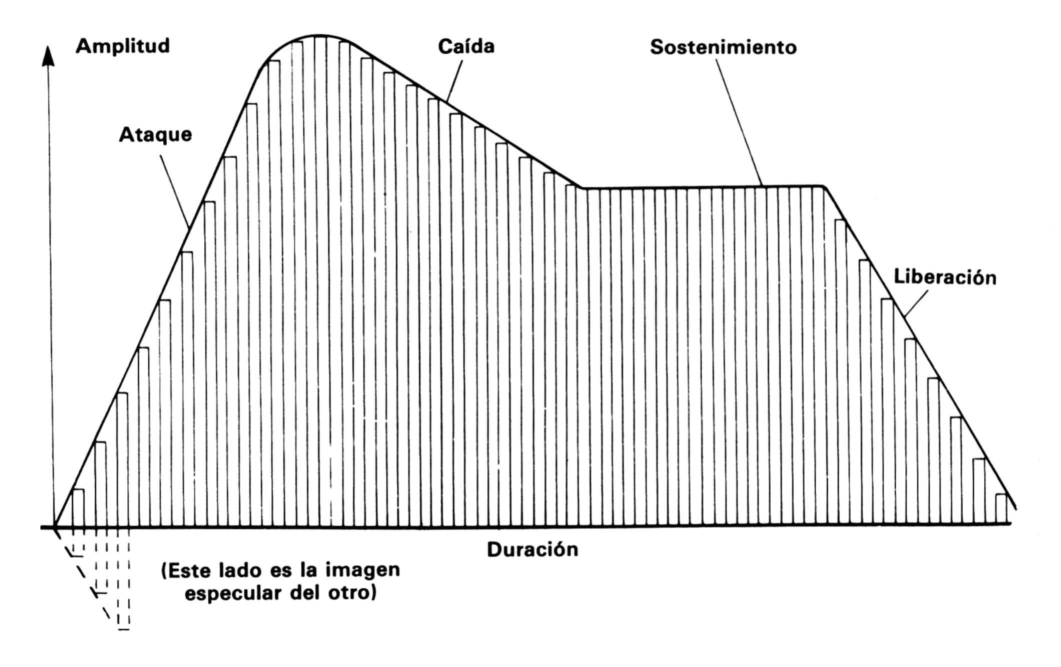


Fig. 11.19 Envolvente de volumen típica que muestra cómo varía la amplitud de una nota con el tiempo.

Por otra parte, el tono de una nota no es constante. Si observa un violinista en acción, verá cómo hace vibrar la mano son la que sujeta las cuerdas. Esto produce un efecto de variación del tono que se denomina «trémolo». Una nota con trémolo suena más real que un simple tono puro. El trémolo y las demás variaciones de frecuencia hacen que la nota tenga «envolvente de tono».

En la figura 11.19 se muestra una envolvente de volumen típica de muchas notas

musicales. Al principio la amplitud crece muy deprisa; es la zona que denominamos «ataque». A continuación viene la de «caída», en la que la amplitud se reduce. Después permanece aproximadamente constante durante algún tiempo (zona de «sostenimiento») para luego decrecer gradualmente hasta cero (zona de «liberación»). El CPC464 nos ofrece la posibilidad de sintetizar una forma de onda tan compleja como ésta mediante una versión ampliada de la instrucción SOUND. La idea es utilizar una instrucción que pueda generar varias formas de envolvente. Cada una de esas formas se puede numerar, y su número se puede especificar en la instrucción SOUND, en su quinto parámetro. Si se utiliza una envolvente, ésta debe controlar la duración de la nota; por eso, cuando se especifica duración 0, el control pasa a ser ejercido por la envolvente. El número de volumen especifica la amplitud inicial de la nota, pero a continuación es la envolvente la que controla el sonido.

ENVOLVENTES

Para crear una envolvente de volumen se utiliza la instrucción ENV, seguida de un mínimo de 4 números o un máximo de 16. Esto parece impresionante a primera vista, pero vale la pena dedicar un esfuerzo a profundizar en los detalles de este magnífico sistema de sonido. Ilustraremos las ideas con un ejemplo. Lo primero que necesitamos es una especie de plantilla, semejante a la de la figura 11.20. El eje vertical

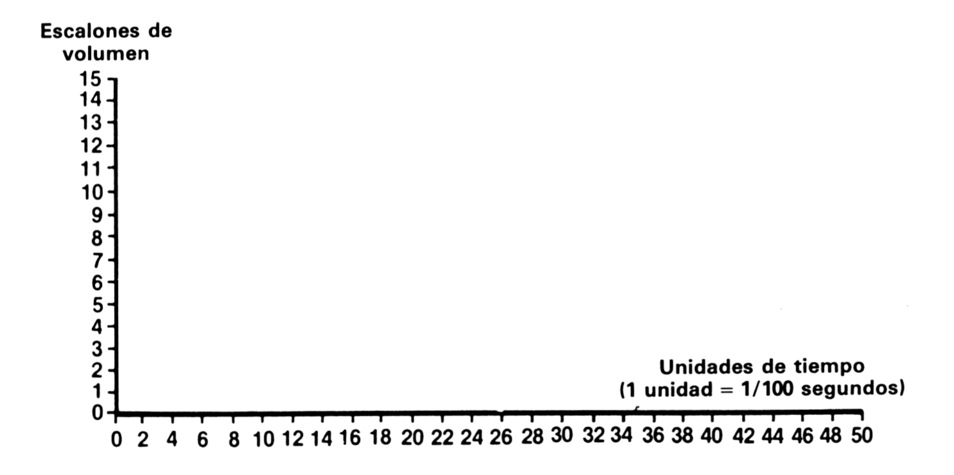


Fig. 11.20 Plantilla para la planificación de la instrucción ENV. El lector puede dibujar estas plantillas con papel milimetrado.

está marcado en escalones de volumen, numerados del 0 al 15, que es el margen permitido para esta instrucción. El eje horizontal está marcado en unidades de tiempo, de 0 a 50. Cada unidad dura 1/100 segundos, luego la duración máxima de las envol-

ventes que se pueden diseñar con esta plantilla es medio segundo. La idea es dibujar sobre la plantilla la envolvente deseada, y luego hacerla coincidir lo mejor posible con valores enteros de los escalones de volumen y de las unidades de tiempo. Puesto que, como decimos, estos valores tienen que ser enteros, toda recta que no sea vertical ni horizontal tendrá que ser simulada en forma de escalera. Lo único que hay que decidir es cuántos escalones se van a utilizar; cuantos más, mejor será la aproximación, pero el máximo es 16.

El ejemplo más sencillo posible es el de una envolvente que decrece uniformemente hacia cero (línea de trazo grueso en la figura 11.21). Para no complicar demasiado

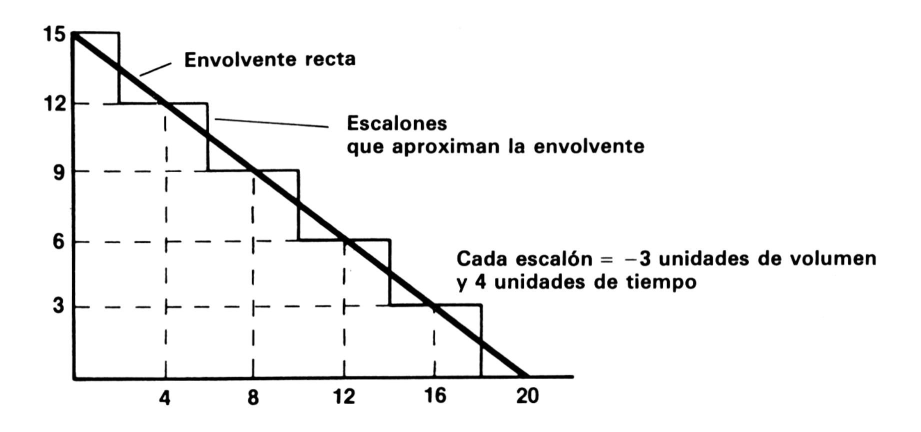


Fig. 11.21 Una envolvente sencilla.

el ejemplo vamos a utilizar solamente cinco escalones, cada uno de altura 3 y de longitud 4. Demos a esta envolvente el número 1. Para programarla, escribiremos ENV seguido del número de la envolvente, que es el 1, y luego el número de escalones y la altura y duración de cada escalón, es decir, ENV 1,5, – 3,4. Ahora ponemos esta instrucción en un programa y la escuchamos (Fig. 11.22).

El programa especifica la envolvente, ENV 1,5, – 3,4, y luego una instrucción

```
10 CLS
20 ENV 1,5,-3,4
30 PRINT"NUMERO DE DURACION CERO"
40 SOUND 2,239,0,15,1,0,0
50 FOR N=1 TO 2000:NEXT
60 PRINT"NUMERO DE DURACION =-10"
70 SOUND 2,239,-10,15,1,0,0
```

Fig. 11.22 Programación de la envolvente.

SOUND que la utiliza. El número de canal elegido es el 2; el número de tono es el 239. El número de duración es 0, y por eso es la envolvente la que controla la duración. Nuestro diseño de la envolvente ha ocupado 18 unidades, luego la duración total del sonido es 18/50 segundos. El volumen inicial es el máximo, 15. El margen para este parámetro es de 0 a 15 cuando se utiliza envolvente. Después del número de volumen viene el de envolvente, 1. La envolvente de tono es 0, porque aún no la hemos definido. Todos estos datos están incluidos en la instrucción de la línea 40. Escuche el efecto al ejecutar el programa. La línea 70 repite el sonido, esta vez poniendo un número negativo como número de duración. El efecto es repetir la envolvente tantas veces como indique ese número (ignorando el signo). Por ejemplo, el número de duración −10 hace que la envolvente se repita 10 veces.

Aun con esta envolvente tan sencilla, de solamente cuatro parámetros, hay mucho que experimentar. Por ejemplo, ponga un 1 como número de tono y un 12 como número de ruido. El efecto es el de un disparo. Hay muchas variaciones posibles, así como mezclas de tonos y ruido. Para conseguir efectos más refinados se pueden diseñar envolventes con hasta *cinco secciones*. Además, los escalones pueden ser de amplitud menor que los del ejemplo anterior; los mejores resultados se consiguen con el máximo número de escalones posible, 15. Probemos con una envolvente más ambiciosa, compuesta por cuatro secciones: ataque, caída, sostenimiento y liberación.

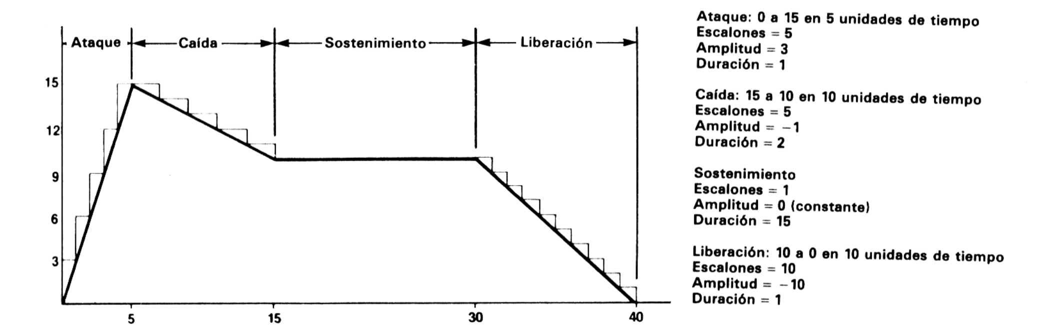


Fig. 11.23 Diseño de una envolvente de cuatro secciones.

El diseño está ilustrado en la figura 11.23. La línea de trazo grueso es la envolvente deseada; la de trazo fino es la aproximación que vamos a realizar. Necesitaremos una instrucción ENV con cuatro secciones. Los parámetros que utilizaremos son los que se indican a la derecha de la figura. El programa correspondiente es el de la figura 11.24. La instrucción ENV especifica en primer lugar el número de la envolvente, y luego una lista de números de escalones y amplitudes y duraciones de éstos, todo ello para cada una de las cuatro secciones. En la instrucción SOUND ponemos el número de duración −10 para repetir el sonido 10 veces. En las líneas 40 a 70 hemos tratado de programar una musiquilla de reloj. Ha salido algo rápida, pero eso tiene fácil remedio: añadir una sección de silencio a la envolvente. Ponga los números 1,0,40 al final de la instrucción ENV y escuche la diferencia.

```
10 ENV 1,5,3,1,5,-1,2,1,0,15,10,-1,1
20 SOUND 2,478,-10,0,1,0,0
30 FOR N=1 TO 2000:NEXT
40 FOR J=1 TO 8
50 READ NOTA%
60 SOUND 2,NOTA%,0,0,1,0,0
70 NEXT
100DATA 478,379,426,638,638,426,379,478
```

Fig. 11.24 Programa que utiliza la envolvente de la figura 11.23.

Para trabajar con estas envolventes hace falta planificación, buen oído y mucha práctica. Aunque las reglas referentes a los parámetros de la instrucción ENV son sencillas, mucho más que en otros ordenadores, unas cuantas sugerencias le facilitarán la tarea. Procure que los escalones sean de amplitud razonable. Si la duración de la envolvente es demasiado pequeña, no habrá mucho que escuchar, a menos que esté programando disparos de pistola. Para notas musicales las envolventes muy

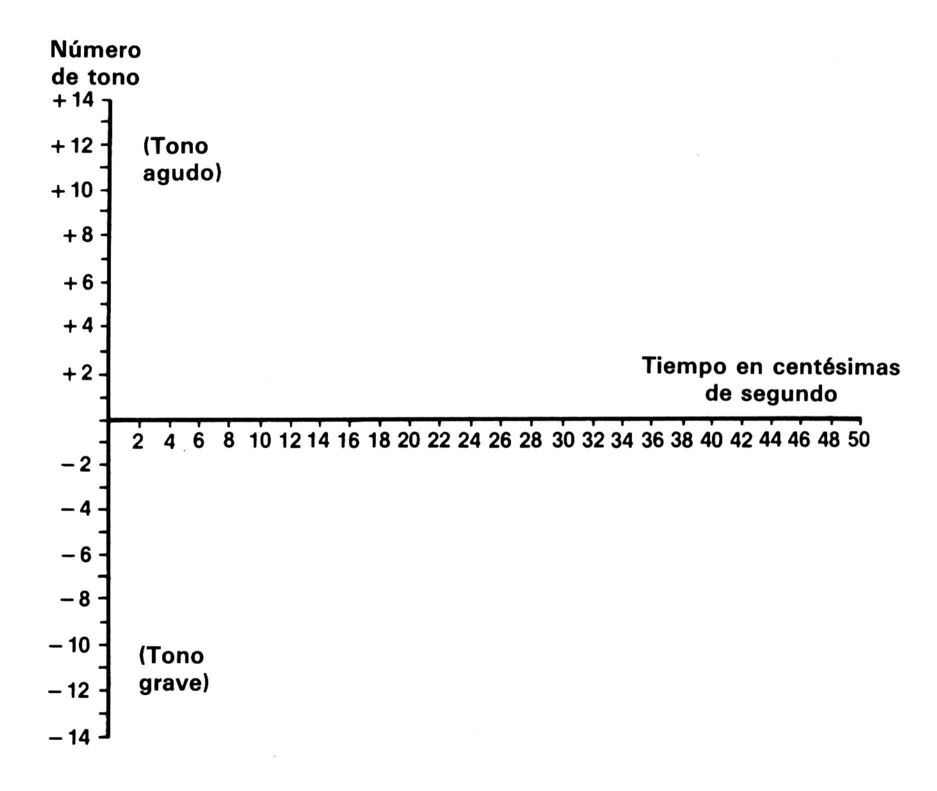


Fig. 11.25 Plantilla para el diseño de envolventes de tono.

cortas producen un efecto desagradable. En general, procure que las notas musicales duren al menos un segundo. Otra sugerencia importante: trate de encajar las variaciones de amplitud en las diagonales de los cuadrados de la plantilla; así los parámetros serán números enteros sin requerir más aproximaciones.

NOTAS CON TRÉMOLO

Ya sabemos que podemos especificar, no sólo envolventes de volumen, sino también de tono. Las envolventes de tono se programan de forma muy parecida a las de volumen. La figura 11.25 muestra una plantilla adecuada. También en este caso la forma de envolvente deseada se aproximará mediante una sucesión de escalones. En el eje vertical tendremos ahora números de tono. Un aumento del número de tono representa una disminución de la frecuencia. Veamos el ejemplo de la figura 11.26, en

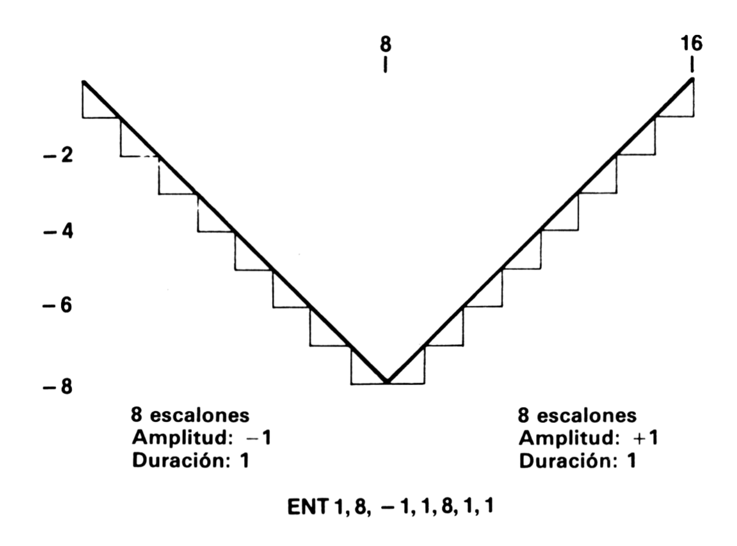


Fig. 11.26 Diseño de una envolvente de tono.

la que hemos diseñado una envolvente de tono muy sencilla. Esta envolvente hace que la frecuencia aumente durante 8/100 segundos y que luego vuelva a decrecer en los 8/100 segundos siguientes. Haciendo coincidir la envolvente con las diagonales de los cuadrados se facilita la lectura de los parámetros. En la figura 11.27 se da el programa correspondiente a esta envolvente.

Con las notas musicales, el efecto trémolo se consigue haciendo subir y bajar el número de tono. El trémolo que se ilustra en la figura 11.28 es bastante lento. En muchos casos es preferible una variación más rápida. En el programa de la figura 11.29 la instrucción ENT contiene algo nuevo. En efecto, el número de la envolvente

```
10 ENT 1,8,-1,1,8,1,1
20 SOUND 2,478,50,7,0,1,0
30 FOR N=1 TO 2000:NEXT
40 FOR J=1 TO 8
50 READ NOTE%
60 SOUND 2,NOTE%,40,7,0,1,0
70 NEXT
100 DATA 478,379,426,638,638,426,379,478
```

Fig. 11.27 Programación de la envolvente de tono de la figura 11.26.

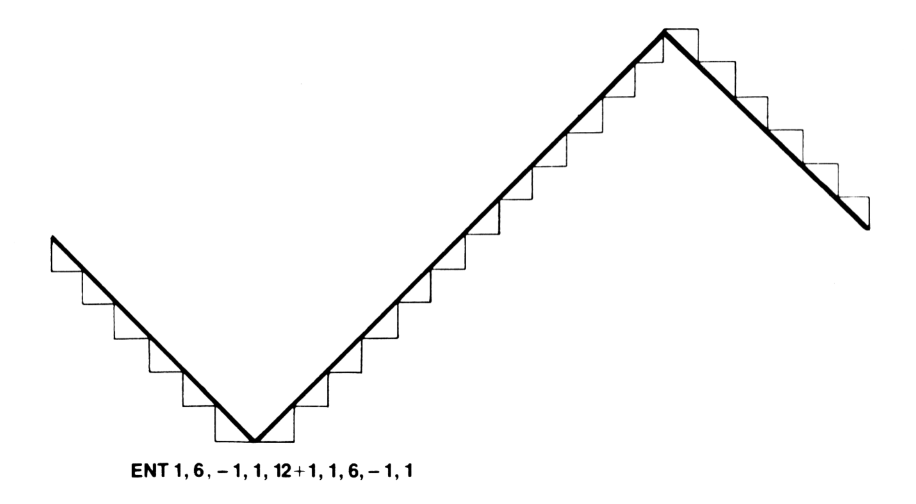


Fig. 11.28 Envolvente de tono para una nota con trémolo.

es –1. El efecto es hacer que la envolvente se repita. Si hubiéramos puesto el 1 sin signo, solamente habría un ciclo de variación del tono; en cambio, al poner el signo menos, el trémolo se repite mientras la nota continúe sonando.

Las combinaciones de ruido con envolventes de tono son tan fascinantes como con envolventes de volumen. Por ejemplo, haga los siguientes cambios en la instrucción SOUND de la figura 11.29: ponga 1 como número de tono, 1200 como duración y 7 como número de ruido. Ejecute el programa y podrá oir una auténtica locomotora de vapor a toda velocidad. Por otra parte, se pueden cominar los dos tipos de envolventes, con lo que prácticamente no hay sonido que no pueda ser generado con el CPC464. Es muy importante idear un método sistemático para la investigación de los sonidos. Grabe siempre las notas que obtenga y archive los diseños de las envolventes. De esta forma llegará a tener una biblioteca de efectos sonoros que puede utilizar cuando los necesite. No olvide que hay revistas ansiosas por recibir colaboraciones y, sobre todo, que algunas pagan por ellas.

```
10 ENT -1,6,-1,1,12,1,1,6,-1;1
20 SOUND 2,478,200,7,0,1,0
```

Fig. 11.29 Programación del trémolo. ENT – 1 hace que la envolvente se repita.

SINCRONIZACIÓN

Se nos acaba el espacio. Para hacer justicia al sistema de sonido del CPC464 necesitaríamos al menos otro libro como éste. Hemos tenido que omitir algunas cosas; en particular, los detalles de la instrucción SQ. No obstante, vamos a dedicar esta última sección al tema de la *sincronización*. Cuando enviamos una sucesión de notas a la cola de sonido de los diversos canales, es muy difícil asegurar que todas ellas sean ejecutadas en el momento adecuado. De hecho, ésta es una tarea imposible en algunos ordenadores. Por ejemplo, de puede necesitar que dos notas suenen simultáneamente en dos canales, o que haya silencio en un canal mientras suena una nota en otro. El CPC464 dispone de un sistema de números de sincronización que controla esas funciones. La sincronización se realiza utilizando los números de selección de canal (Fig. 11.30). Además de los números que hacen que se envíen señales a los tres canales, hay tres que se encargan de la sincronización, uno que hace que una nota quede espeando en la cola y otro que borra el contenido de las colas; este último proporciona un método rápido para hacer silencio. Se pueden conseguir varios de estos efectos sumando estos números.

Número de selección	Efecto
8	Sincronizar con el canal A
16	Sincronizar con el canal B
32	Sincronizar con el canal C
64	Retener
128	Borrar los tampones de las colas de sonido

Fig. 11.30 Conjunto completo de números de selección de canal, incluyendo los de sincronización.

Consideremos el ejemplo de la figura 11.31. La línea 10 contiene una instrucción SOUND. Si el número de selección de canal fuera alguno de los que hemos utilizado hasta ahora, la instrucción generaría inmediatamente un sonido. Pero el número 33 lo impide. Este número es la suma de 1, que es el que selecciona el canal A, más 32, que obliga a sincronizarlo con el C. Por lo tanto, el sonido queda retenido en la cola y no se oirá nada mientras no empiece a sonar el canal C. Tras una pausa llegamos a la instrucción que afecta al canal C. El número de selección de canal es ahora 12, igual a 4 (número del canal C) más 8 (sincronizar con el A). Obsérvese

```
10 SOUND 33,478,300
20 FOR N=1 TO 2000:NEXT
30 SOUND 12,379,300
```

Fig. 11.31 Un ejemplo de sincronización.

que *los dos números de selección de canal han de contener números de sincronización*. El efecto global del programa es que no suena nada hasta el final de la pausa, y que entonces se oye el acorde.

Si todo esto le desanima, consuélese pensando que se pueden conseguir muchos efectos de forma más sencilla. Puede hacer que el programa lea los números de selección de canal y los números de tono de una lista da DATA, y poner un 0 para los silencios. Si no le apasiona la composición musical, puede ignorar los problemas de sincronización. No obstante, disponer de estos recursos en un ordenador de este precio es una suerte; como todos los talentos, éste no se marchita por no utilizarlo.

Apéndice A

Edición

En este contexto, *editar* significa modificar algo que ha aparecido en la pantalla. Como ya sabemos, se puede borrar un carácter con la tecla DEL, o volver a escribir la línea entera. También se puede borrar el grupo de líneas con la orden DELETE (por ejemplo, DELETE 10-100). Pero editar consiste en modificar solamente lo erróneo de una línea, sin alterar lo demás. Todo el contenido de una línea, incluso su número, se puede modificar por edición. El proceso de edición se puede realizar:

a) cuando se está tecleando la línea, antes de pulsar ENTER;
b) después de introducir la línea, pero antes ejecutar el programa;
c) cuando el ordenador detecte un error durante la ejecución del programa.

Analicemos las tres posibilidades:

a) Durante la introducción de una línea se pueden utilizar todos los recursos de edición de línea. La edición termina cuando se pulsa ENTER.
b) Cuando ya se ha introducido la línea se puede utilizar cualquiera de los métodos de edición que describimos más abajo.
c) Cuando el programa se detiene con un mensaje de error, la línea en que se ha detectado el error puede (dependiendo del tipo de error) aparecer en la pantalla, con el cursor situado sobre el número de línea. El error se puede corregir por el método de edición de línea.

MÉTODOS DE EDICIÓN

Una característica peculiar del CPC464 es que la edición se puede realizar por dos métodos distintos. La mayor parte de los ordenadores utiliza, o bien edición de línea, o bien edición de pantalla. La edición de línea consiste en traer a la pantalla la línea que se quiere modificar, para lo cual se utiliza una orden tal como EDIT 200 (para editar la línea 200). El cursor se puede desplazar entonces sobre la línea para corregir el error. En la edición de pantalla, cualquier línea visible en la pantalla se puede corregir sin más que llevar el cursor hasta ella para hacer la modificación. Los dos métodos tienen partidarios y detractores acérrimos; el CPC464 satisfará a todos.

ÓRDENES DE EDICIÓN

1. *Edición de línea*

La línea, con el cursor sobre su número, puede estar ya en la pantalla por haber detectado el ordenador un error en ella. Si no es así, teclee EDIT seguido del número de línea que quiera corregir (dejando un espacio entre la T y el número). La línea completa aparecerá en la pantalla, con el cursor situado sobre el primer dígito del númeo de línea. Ahora se pueden utilizar las cuatro teclas de movimiento del cursor para llevarlo al lugar deseado. El carácter que está debajo del cursor se borra pulsando CLR. Si se mantiene pulsada esta tecla se borra rápidamente el texto que está a la derecha del cursor. Análogamente se puede borrar hacia la izquierda del cursor con la tecla DEL. Cuando se teclee un carácter éste es insertado en la posición del cursor. Para terminar la edición pulse ENTER. También se puede editar una línea por este procedimiento cuando se la está tecleando por primera vez. Por ejemplo, si ha escrito

PRINT ESTO ES EL FIN''

y se da cuenta de que le faltan las primeras comillas, haga retroceder el cursor hasta situarlo sobre la E de ESTO, escriba las comillas y pulse ENTER. La tecla ENTER se puede pulsar en cuanto se haya corregido el error: no es necesario llevarlo hasta el final y de la línea.

2. *Edición por copia*

La edición por copia es la versión particular del CPC464 de la edición de pantalla. Este método permite copiar una línea, omitiendo o reemplazando parte de ella. Se puede editar por copia cualquier línea visible en la pantalla, incluso las órdenes directas. Mantenga pulsada la tecla SHIFT y utilice las teclas de movimiento del cursor para situarlo sobre la línea que quiera editar. Suelte ahora la tecla SHIFT y las del cursor, y pulse la tecla COPY para copiar la línea o parte de ella. La copia empieza a partir de la posición en la que dejó el cursor. Si quiere saltarse parte de la línea, pulse la tecla de cursor a la derecha en lugar de COPY. Pulse ENTER cuando haya terminado de copiar *todo* lo que quería. Si quiere copiar la línea completa, hágalo antes de pulsar ENTER, porque esto concluye el proceso de copia. Ésta es una diferencia importante entre la edición de línea y la edición por copia.

Este método es muy versátil. Por ejemplo, se puede cambiar un número de línea. Supongamos que usted tiene en la pantalla la línea 200 y quiere convertirla en la 1000. Escriba el número 1000; con la tecla SHIFT pulsada, lleve el cursor hasta el principio de las instrucciones de la línea 200 (no al número de línea). Copie la línea con la tecla COPY y pulse ENTER. En este momento ha quedado introducida la línea 1000 en el programa. Para suprimir la 200, escriba este número y pulse ENTER.

También se puede copiar texto que no haya tenido número de líneas antes. Supongamos que usted está introduciendo un programa y ha escrito FOR N = 1 TO 200:NEXT, olvidando el número de línea. No es necesario que vuelva a escribir la instrucción entera. Escriba el número de línea correcto; lleve el cursor (con SHIFT

pulsada) hasta situarlo sobre la F de FOR, copie la instrución con COPY y pulse ENTER. Este método permite también copiar parte de una línea para introducirla en otra. Supongamos, por ejemplo, que en la línea 400 hay una instrucción que queremos poner también en la 700. Haga que ambas líneas aparezcan en la pantalla (quizá con LIST 400 y LIST 700). Copie la línea 700 hasta el lugar donde quiera insertar la instrucción; lleve el cursor hasta el principio de la instrucción en la línea 400; copie la instrucción y luego vuelva con el cursor a la línea 700 para terminar de copiarla. Pulse ENTER cuando la línea nueva esté a su gusto.

DETECCIÓN Y CORRECCIÓN DE ERRORES

Un programa puede contener errores de diversos tipos, no todos detectables por la máquina. El ordenador informa de los que encuentra con mensajes generalmente obvios. Por ejemplo, «Line does not exist» (la línea no existe) significa que se ha programado GOTO 1000 o GOSUB 1000 y se ha olvidado introducir la línea 1000. También puede aparecer este mensaje de error si se ha intentado borrar la línea 1000 sin que existiera, o si se ha programado THEN 1000 ELSE 2000 detrás de algún IF.

El error más frecuente es el de sintaxis, «Syntax error». Este error consite en una mala utilización de las palabras reservadas de BASIC. Puede tratarse de un error mecanográfico: PRIBT en lugar de PRINT. Puede faltar un paréntesis, una coma, un punto y coma. Se puede haber escrito un punto y coma en lugar de dos puntos.

Otro error frecuente es «Improper argument» (argumento incorrecto). Normalmente se produce porque el argumento de una función tiene un valor fuera del margen permisible. Un ejemplo es TAB(300). Por supuesto, después de leer este libro usted no pondría TAB(300) en un programa, pero posiblemente sí TAB(N), y N podría llegar a tomar el valor 300 de forma inesperada. Toda función que requiera argumento (como MID$, LEFT$, RIGHT$, INSTR, STRING$, etc.), puede dar lugar a este error. Lo mismo ocurrirá si N es negativo en SQR(N), o si N es negativo o cero en LOG(N), o como resultado de otras operaciones matemáticas imposibles. La causa no debería ser difícil de encontrar, ya que el ordenador indica en qué ha encontrado el error.

No se sorprenda por el elevado número de errores que pueden «colarse» en un programa, aun cuando lo esté copiando de una revista. En general, los programas que se publican en las revistas son muy fiables, pero algunos están impresos de forma tal que fomentan la introducción de errores. A veces se confunden unas letras con otras: por ejemplo, la I (i mayúscula) con la l (L minúscula), o el número 0 con la letra O mayúscula. Una línea tal como

```
IFM=10ORJ=4ORD=2
```

puede ocasionar confusión. Hay una revista concreta que parece estar especializada en líneas como ésta. Si se hubiera escrito

```
IF M=10 OR J=4 OR D=2
```

se habría evitado el problema. A veces es necesario este ahorro de espacio para que el programa quepa en la memoria del ordenador, pero ésta es la única excusa válida.

Incluso después de depurar todos los errores de sintaxis y de argumentos incorrectos, puede ocurrir que el programa no funcione tal como se esperaba de él. El CPC464 ofrece todo lo que se puede esperar de una máquina moderna en cuanto a medios de detección de errores. Uno de los más eficaces es la tecla ESC. Como el lector ya sabe, al pulsar esta tecla una vez se detiene el programa, y al pulsar cualquier otra tecla la ejecución continúa. Como veremos más adelante, esto es muy útil en la detección de errores en programas gráficos; en cambio, en programas de otro tipo puede ser más útil pulsar ESC dos veces, con lo que el programa se detiene y en la pantalla aparece el número de la línea en la que se ha interrumpido la ejecución. Lo que quizá el lector no sepa es que se puede comprobar el valor de las variables, e incluso cambiar su valor, cuando el programa está detenido; la ejecución se reanuda luego con una instrucción GOTO, o bien, si no se ha modificado el programa, con la orden CONT. Supongamos que estamos ejecutando un programa que tiene un contador lento y que se pulsa ESC dos veces al principio de la ejecución. La ejecución del programa se detiene con el mensaje «Break in 40» (interrupción en 40). Supongamos que la variable del contador es N. Teclee ?N y pulse ENTER: el valor actual de N aparecerá en la pantalla. Para comprobar qué ocurrirá al final del bucle, teclee N = 998 (ENTER). Para continuar teclee CONT (ENTER); si durante la interrupción ha modificado o suprimido alguna línea del programa o añadido líneas nuevas, el ordenador responderá con el menaje «Cannot CONTinue» (no puedo continuar); en tal caso, para reanudar la ejecución del programa en el punto en que fue interrumpida tendrá que teclear GOTO 40 (ENTER). Ésta es una buena forma de comprobar el funcionamiento de los bucles, pues ahorra el tiempo que se perdería si hubiera que esperar a que el contador llegara al final por sí mismo. Incluso se puede automatizar el proceso con la instrucción ON BREAK GOSUB: al pulsar dos veces ESC se ejecuta la rutina especificada, y se puede hacer que ésta escriba los valores de las variables de interés o que capte los valores nuevos.

La tecla ESC, pulsada una vez en lugar de dos, es útil en la depuración de programas gráficos. Cada vez que se pulsa ESC se congela la acción de los gráficos (aunque no el parpadeo de las tintas), por lo que esta tecla se puede utilizar para observar en qué orden se desarrollan las cosas. Al pulsar cualquier otra tecla se reanuda la ejecución. El programa se puede interrumpir por este procecimiento cuantas veces se desee. Si con esto no basta, introduzca en el programa retardos provisionales para observar la acción en cámara lenta; utilice la tecla ESC en los pasajes más delicados.

CÓMO SEGUIRLE LA PISTA A LOS BUCLES

El programador puede sentirse perdido cuando el programa funciona incorrectamente sin emitir mensajes de error. Esto puede ser un signo de falta de planificación o un simple descuido. El método de seguimiento con ON BREAK GOSUB puede ser útil, pues permite la automatización de labores tales como escribir los valores de las variables y reanudar el programa. Pero a veces es necesario un método de seguimiento más drástico. Cuando un programa contiene muchas instrucciones IF ... THEN ... ELSE, puede ocurrir que alguna de ellas no funcione como se pensaba. Para estos casos el CPC464 dispone de otras dos instrucciones: TRON y TROFF.

TRON escribe en la pantalla los números de las líneas a medida que éstas son ejecutadas. Los números aparecen entre corchetes, antes de los textos escritos por las instrucciones PRINT de la línea correspondiente. Compruebe el efecto de esta instrucción con un programa lento. Recuerde que se puede combinar TRON con los demás métodos de detección y corrección de errores. TROFF anula el efecto de TRON.

Apéndice B

Conexión de una impresora

El CPC464 es extraordinariamente fácil de conectar a cualquiera de las impresoras populares dotadas de interfaz en paralelo tipo Centronics. Además de la impresora se necesita el cable adecuado, que el lector puede adquirir en la tienda en que compró el ordenador. Un extremo del cable se conecta a la salida PRINTER del ordenador; el otro, al zócalo de la impresora. La variedad de impresoras disponibles es tan grande que no vamos a hacer sugerencias específicas: desde los modelos más económicos hasta las impresoras de margarita, pasando por las impresoras gráficas que permiten hacer «volcados de pantalla». Lo cierto es que con el CPC464 no se está limitado a las impresoras de la misma marca, como ocurre con otros ordenadores, sino que se puede conectar cualquier impresora que utilice un interfaz paralelo Centronics.

Según el manual, el CPC464 envía a la impresora dos códigos al final de cada línea. Uno es el de avance de línea, ASCII 10, y el otro el de retorno del carro, ASCII 13. Casi todas las impresoras populares se pueden ajustar para que avancen el papel cuando reciban *cualquiera* de estos códigos; el ajuste se realiza normalmente con un conmutador interno de la impresora. Si al lector le ocurre que la impresora lo escribe todo en la misma línea o que salta dos líneas en lugar de una, el remedio está en cambiar la posición del conmutador. Yo he observado que todas las impresoras que he conectado con el CPC464 saltan dos líneas, lo que indica que el ordenador envía a la impresora dos códigos de avance de línea. En algunas impresoras esto se resuelve seleccionando la mínima interlínea, 7/72 pulgadas. Para el caso concreto de la Epson RX-80, la orden necesaria es

```
PRINT*8,CHR$(27);"A";CHR$(7)
```

que se debe enviar a la impresora cuando ésta se encuentra encendida. Por otra parte, es necesario que el número de caracteres por línea no sea menor en la impresora que en el ordenador. Por ejemplo, si la impresora es de 40 caracteres por línea, en el ordenador hay que teclear la orden WIDTH 40.

Apéndice C

Programación del teclado

Una característica esencial de los ordenadores modernos es la posibilidad de programar teclas, esto es, de asignar a determinadas teclas ciertas funciones, de modo que posteriormente se ejerza cada función sin más que pulsar la tecla correspondiente. Por ejemplo, para no tener que teclear LIST y luego pulsar ENTER cada vez que se quiera listar un programa, se puede programar una tecla que realice este trabajo con una sola pulsación. Otra tecla se podría programar con PRINT TAB(2)", etc.

Algunos ordenadores tienen teclas especiales programables, pero en muy pocos la programación es sencilla. El CPC464 permite definir hasta 32 teclas, utilizando para ello instrucciones BASIC. No obstante, hay algunas restricciones. A cada tecla no se puede asignar más de 32 códigos. El número total de códigos para todas las teclas programadas no puede exceder de 120. Pero en la práctica no es de esperar que estas limitaciones sean importantes.

Supongamos, pues, que deseamos redefinir una tecla. Parece lógico elegir una que no se utilice normalmente. Para este propósito se puede utilizar los códigos ASCII del 128 al 159. Los trece primeros de estos códigos ya están asignados a las teclas del teclado numérico. En el apéndice III, página 15, del manual se dan los códigos asignados a estas teclas. Por ejemplo, el código de la tecla del 0 (en el teclado numérico) es el 128. Para programar esta tecla con la función LIST (ENTER) se teclea

KEY 128,"LIST" + CHR$(13)

y luego se pulsa ENTER. En la definición de teclas, CHR$(13) significa ENTER. A partir de ahora, cada vez que se pulse la tecla del 0 del teclado numérico aparecerá en la pantalla el listado del programa.

Tomemos ahora un código ASCII que no esté asignado a ninguna tecla, por ejemplo el 156. Definamos

KEY 156,"PRINT:PRINT" + CHR$(13)

Una vez definido el carácter, vamos a asignarlo a alguna tecla, por ejemplo, a la de «abrir corchetes». Teclee

KEY DEF 17,1,156

y pulse ENTER. El número 17 en esta orden es el número de la tecla; el 1 indica

que la tecla debe ser de repetición; el 156 es, evidentemente, el nuevo código ASCII que se le ha asignado. Todas las teclas pueden ser redefinidas de esta forma. Las definiciones se pierden cuando se reinicializa la máquina con SHIFT CTRL ESC. Si no quiere que la función asignada a una tecla sea ejecutada inmediatamente, omita el CHR$(13) en la definición. Por ejemplo, se podría programar una tecla con PRINT TAB(, para luego completar el número de tabulador, el paréntesis y el texto.

Apéndice D

Trampas para errores

En este libro ya hemos hablado de las trampas para errores. Como hemos visto, se trata de mecanismos con los que se validan los datos que se introducen por el teclado. Una trampa típica es del siguiente estilo:

60 IF LEN(A$) = 0 THEN GOTO 1000:GOTO 50

Es necesario programar una trampa para cada tipo de error posible. Esto puede ser muy laborioso, y al final siempre se produce un error en el que no se había pensado. El CPC464 es una de las pocas máquinas que disponen de la instrucción ON ERROR GOTO. En la figura D.1 damos un ejemplo (bastante artificial, desde luego, pero un ejemplo tomado de la vida real sería demasiado largo). El programa mide la longitud de una palabra y calcula su inverso. La división es imposible cuando la longitud de la palabra es nula; cuando esto ocurre entra en acción la instrucción ON ERROR GOTO. La palabra carecerá de letras si se pulsa ENTER sin haber pulsado antes ninguna otra tecla. Normalmente, si esto ocurriera el programa se detendría con un mensaje de error. El efecto de ON ERROR GOTO es evitar que el programa se detenga en caso de error y reconducirlo hacia la rutina especificada. En este caso la rutina escribe un mensaje y reanuda (RESUME) la ejecución en la línea 20. El inconveniente de este método es que la rutina se ejecuta como resultado de cualquier error, por ejemplo, de un error de sintaxis.

```
10 ON ERROR GOTO 1000
20 PRINT"ESCRIBA UNA PALABRA"
30 INPUT A$
40 L=LEN(A$)
50 PRINT 1/L
60 END
1000 PRINT"LA PALABRA NO TIENE NINGUNA LETRA!"
1010 RESUME 20
```

Fig. D.1 Utilización de ON ERROR GOTO (en caso de error ir a ...).

En la figura D.2 damos otro ejemplo. La línea 20 asegura que el programa salte, en caso de error, a la línea 1000. El bucle que empieza en la línea 30 lee un conjunto de números, calcula su raíz cuadrada y la convierte en una cadena literal. Ahora bien, uno de los números es negativo, luego el ordenador no puede calcular su raíz cuadrada, pues ésta es un número «imaginario»: al elevar al cuadrado un número,

```
10 CLS
20 ON ERROR GOTO 1000
30 FOR N=1 TO 5
40 READ X:Y$=STR$(SQR(X))
50 PRINT"NUMERO";X;"  RAIZ CUADRADA:";Y$
60 NEXT
70 DATA 5,4,3,-2,2
80 END
1000 Y$=STR$(SQR(ABS(X)))+"J"
1010 RESUME 50
```

Fig. D.2 Otro ejemplo de trampa para errores con reanudación automática de la ejecución del programa.

tanto si es positivo como si es negativo, el resultado es siempre positivo; por consiguiente, no hay ningún número real cuyo cuadrado sea negativo. No obstante, en algunas aplicaciones atribuimos un sentido a las raíces de los números negativos. Cuando el programa lee el −2, al llegar a la línea 40 se produce un error y el programa salta a la línea 1000. En ésta se obtiene el valor absoluto de −2, se calcula su raíz cuadrada y se añade una «J» a la cadena para indicar que el cuadrado era negativo. La línea 1010 hace que se reanude la ejecución en la línea 50.

```
10 CLS
20 ON ERROR GOTO 1000
30 FOR N=1 TO 5
40 READ X:Y$=STR$(SQR(X))
50 PRINT"NUMERO";X;"  RAIZ CUADRADA";Y$
60 NEXT
70 DATA 5,4,3,-2,2
80 END
1000 Y$=STR$(SQR(ABS(X)))+"J"
1005 PRINT"LINEA";ERL;" -- ERROR";ERR
1010 RESUME 50
```

Fig. D.3 La subrutina escribe los números de línea y de error.

En el programa de la figura D.3 se introduce una modificación que permite conocer qué error se ha producido y en qué línea. En 1005, ERL es el número de la línea en la que se ha detectado el error; EPR es el código del error. (Al final del manual del CPC464 hay una lista de errores que se debe tener a mano cuando se está desarrollando un programa.) Esta línea es muy útil cuando el error no es obvio.

PRINT USING

Hay una última instrucción, PRINT USING, de la que no hemos podido hablar antes porque es demasiado complicada. El objeto de la instrucción es crear una especie de plantilla para la escritura. Después de la palabra USING debe venir una cadena literal, que puede haber sido especificada en alguna línea anterior del programa. La forma de esa cadena es la que impone cómo se va a presentar el texto en la pantalla.

```
10 CLS
20 PRINT TAB(15)"PRINT USING"
30 PRINT:PRINT
40 N=140.2716
50 PRINT"N ES";N
60 A$="###.##"
70 PRINT"CON PRINT USING, N ES";USING A$;N
80 PRINT"EL IVA SOBRE $";N;"ES";15*N/100
90 PRINT"QUEDA MEJOR SI ESCRIBIMOS $";USING A$;N*
   15/100
```

Fig. D.4 Ejemplo de utilización de PRINT USING.

Como esto suena a misterio, veamos el ejemplo de la figura D.4. En este ejemplo hemos definido A$ como # # # . # #. Esto significa que los números que se escriban utilizando A$ como plantilla aparecerán con tres dígitos antes del punto decimal y dos después de él. Ésta es la aplicación más importante de PRINT USING, pero el manual describe otras. Algunas de ellas sólo sirven para escribir cantidades en dólares (y el BASIC de esta máquina fue diseñado en el Reino Unido!).

Índice

A

Acorde 153
Actualización de ficheros 105, 106
Adaptador, antena TV 3
Agentes móviles 129
Alfabeto griego 120
Alta resolución 119
Altavoz 9, 145
Amplificador estereofónico 156
Amplitud 147
Animación 123, 124
  de gráficos
    con INK 143
    con TAG 141, 142
Apagador del piano 154
Archivos en cintas 89
Aritmética con cadenas literales 32
Armonía de tres voces 154
ASC 64, 65
ASCII, códigos 57, 58, 64, 65, 66, 67
Asterisco 19
Asterisco parpadeante 77
Ataque 159
AUTO 27, 28

B

Baja resolución 119
Barra inclinada hacia la izquierda 19, 126
Bases de datos 78
BASIC 18
BORDER 110, 111
Borrado de variables 84
Bucles 45
  anidados 47
Búsqueda de ficheros 101-104

C

Cable 3, 4
  de antena 3
Cadena
  literal 20
  vacía 55, 56
Caída 159
Cálculos aritméticos 19, 21
Camino aleatorio 135
Campo 92
Canales 89-91
CAPS LOCK, tecla 9
Cara o cruz 52
Caracteres
  gráficos 120, 121
  no imprimibles 117
Cara o cruz 52
Carga de programas 10
CAT 15
Centrado de títulos 59, 60
CHAIN 15
CHR$ 65
Cinta de demostración 11
Círculos 139
Circunferencias 138, 139
Clavija de red 1, 2
CLOSEIN 97, 98
CLOSEOUT 94, 96
CLS 13
Código de máquina 78
Cola de sonidos 151-153
Color 3, 110-115
Coma 23, 24
Comparación de cadenas 66, 67
Concatenación 33
Conector de seis patillas 1
Contador de cinta 12

Copia de seguridad 7
COPY, tecla 9
Creación de ficheros 94-96
Cronómetros 129
CTRL, tecla 9
Cursor 8
  teclas del 9

D

DATA 36, 37
Decrementar 38
Definición
  de funciones 42, 43
  de variables 29
DEFINT 43, 44
DEFREAL 44
DEFSTR 44
DEL, tecla 9
DIM 68, 69
Dimensionado de listas 69
Diseño
  de programas 78
  de subrutinas 82-84
Dispositivo 89, 90
DRAW 134-139
DRAWR 135, 136
Duración de las notas 148

E

Edición
  por copia 168
  de línea 167
  de pantalla 167
Efectos especiales 155, 156
ELSE 52
ENT 163
ENTER, tecla 9, 10
Envolvente 153, 154, 158-162
  de tono 158
  de volumen 158, 159
EOF 101
Error(es) 169, 170
  de argumento incorrecto 169
  de sintaxis 169, 170
ESC, tecla 9, 45, 46, 170
Escala de do mayor 152
Estéreo 145
EVERY 128, 129
Expresión algebraica 29, 30

F

Fichero(s) 91
  de acceso aleatorio 93
  secuenciales 93
Filas y columnas 22, 24
Finalizador 50
FOR 46, 47
Fracción binaria 41
FRE 117
Frecuencia del sonido 147
Fuente de alimentación 1
Funciones
  literales 57-67
  matemáticas 38-40
Fusible 1

G

GOSUB 73, 75-77
GOTO 45
Grabación de programas 11, 82
Gráfica 132-134
Gráficos 119
  de teclado 120

H

Hertzio 147

I

IF 50, 51
Impresora 21, 172
Incrementar 38
Indicador de fin de fichero 96
INK 112, 113
INKEY 54-56
INKEY$ 55, 56
INPUT 34-36
INSTR 71
Interrupción de bucles 45, 46

L

Lectura de ficheros 99-101
LEFT$ 62, 63
LEN 59, 60
Lenguajes de programación 17
Liberación 159
LINE IMPUT 36
Líneas multi-instrucción 23
Lista 68, 69
Listados 12
LOAD 13
LOCATE 26, 27
LOWER$ 116, 117

M

Magnetófono 4, 5,
Matriz 69, 70
Mayúsculas 8
Menú 73-75
MERGE 15
Mezcla de ruido 156
MID$ 63-65
Minúsculas 8
MOD 42
MODE 27
Modo(s) 112-114, 121
  directo 19
  de programa 17
Monitor 1, 2, 3, 4, 6, 7, 8
MOVE 133
Música 154, 155

N

NEW 13
NEXT 46-51
«NEXT missing», mensaje 47
Nombre
  de fichero 13, 91-93
  de variable 29, 31
Notación científica 41
Número(s)
  de duración negativo 161
  de línea 12, 18, 19
  de selección de canal 150
  de tono 151
    del ruido 156
  de volumen 153

O

Octava 149
ON BREAK 134
ON ERROR 175
Ondas de sonido 147, 148
OPENIN 91
OPENOUT 91
Operaciones prohibidas 32
Orden
  alfabético 67, 68
  de prioridad 39, 40
Órdenes de edición 18
Origen 132
ORIGIN 133
«Overflow», mensaje 42

P

Palabra(s)
  clave 90, 91
  de instrucción 18
  reservadas 18, 30
Pantalla 7, 8
  gráfica 119
  de textos 119
PAPER 112-114
Partición de palabras 22
Pasada por el bucle 46, 47
Pausas en la música 148
PEN 112-114
Pentagrama 148, 149
Pitido 145, 146
Pixel 131
Planificación de gráficos 136, 137
Plantilla
  para el diseño de caracteres 124-128
  para gráficos 121, 122, 131
PLOT 132-134
PLOTR 132
POS 129
Precisión en los cálculos 39-41
Presión del aire 146, 147
«Press any key», mensaje 13
PRINT 21, 23
PRINT USING 177
Programación estructurada 79
Programas protegidos frente a copias 11
Protección 14
Puntero 37

R

RANDOMIZE TIME 140
READ 36, 37
Redefinición de teclas 126, 127, 173, 174
«Redo from start», mensaje 36
Registro 92
REM 11
RENUM 28
Representación de gráficas 132, 133
Resolución 119
Respuesta con una sola tecla 55, 56
RESTORE 65, 66
RETURN 75-77
RIGHT$ 62, 63
Ruido
  de locomotora de vapor 164
  de martillazos 157
  de olas 157
RUN 20

S

SAVE 13
Semitonos 149
SHIFT, teclas 9
Signo «distinto de» 51
Sincronización 142, 165, 166
Sintonía del televisor 7
Sostenimiento 159
SOUND 150-153
SPACE$ 117
SPC 25
SPEED INK 112, 113
SPEED KEY 118
SPEED WRITE 14, 15, 97, 98
SQ 152
STEP 47, 48
STRIGN$ 33
STR$ 60
Subrayado 115, 116
Subrutina 73-76
SYMBOL 126
SYMBOL AFTER 126
«Syntax error», mensaje 10

T

TAB 24, 25
Tabulación 24
TAG 141, 142
Tampón 89-91, 94-96
Teclado 8-10
  numérico 9
Teclas
  negras 149
  programables 173, 174
  repetitivas 10
Televisor 2, 3, 4, 6, 8
TEST 141
Texto parpadeante 115
THEN 50-52
TIME 111
Tinta parpadeante 115, 116
Tomas de corriente 4
Tono de una nota 147-150
Totalizador de números 49, 50
Trampa para errores 54, 175
Trémolo 163-165
TROFF 171
TRON 171

U

«Unexpected RETURN», mensaje 77
UPPER$ 116, 117

V

VAL 60, 61
Valor del contador del bucle 47, 48
Variable
  entera 41, 42
  indexada 68
  literal 30, 32
Velocidad
  del magnetófono 14
  de parapadeo 112
Ventana 107-110
VPOS 129

W

WEND 53-56
WHILE 53-56
WIDTH· 172
WINDOW SWAP 109
WRITE 117

Z

ZONE 24

91

AMSTRAD
Avda.del Mediterráneo, 9. 28007 MADRID

PROGRAMANDO CON AMSTRAD

Ian Sinclair


AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
https://acpc.me/